AD-A044601

1

COBOL COMPILER VALIDATION SYSTEM - 1974

(CCVS74)

Version 3 Release 0 (3.0)

PRODUCED BY:

FEDERAL COBOL COMPILER TESTING SERVICE
SOFTWARE ENGINEERING SERVICES
DEPARTMENT OF THE NAVY
WASHINGTON D.C. 20376

(202) 697-1247

DDC
RECEIVED
NOV 21 1977
D

Report Number FCCTS/CCVS74-77/20

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

UNITED STATES DEPARTMENT OF COMMERCE
National Technical Information Service
5285 Port Royal Road
Springfield, Virginia 22161

Date 11-14-77 NTIS Control # 300023

TO: Defense Documentation Center - DDC/TC
Cameron Station
Alexandria, Virginia 22314

FROM: NTIS, Input Branch
5285 Port Royal Road
Springfield, Virginia 22161

Report # FCCCTS/CCVS74-77-2Q ADA # 046 601

Title: Federal Cobol Compiler Testing Service (Documentation)

Subject report is ☐ Standard Process ☒ STG report. STG - Special Technology Group. ☐ Computer Product. ☐ Follow up date _____.

☐ The report will be accessioned by DDC. The form noting the ADA number is returned.

☐ The report has been assigned the ADA number noted above and is returned to NTIS for processing.

☐ DDC will not process the report. It is returned to NTIS.

Mag Tape Price _____
PC & MF Price $12.50 - $12.50 Source $ ADPES

Stock Quantity _____ Source Share _____

Comments:

AD-A046 601

_____
Signature (for billing)

Copy when completed to
Finance Branch.

DOD Report Action Request
(Replaces NTIS-164 5-72)

Dottie Adams
processor

AMERICAN REVOLUTION BICENTENNIAL 1776-1976

TABLE OF CONTENTS

CCVS74 VERSION 3 * RELEASE 0 77/10/13

COBOL COMPILER VALIDATION SYSTEM - 1974 (CCVS74)

Version 3 Release 0 (3.0)

1. INTRODUCTION

The 1974 COBOL Compiler Validation System is based on the technical specifications contained in American National Standard Programming Language COBOL, X3.23-1974. It is made up of audit routines, their related data and an executive routine (PPLVP74) which manages the audit routines and prepares them for compilation. Each audit routine is a COBOL source program which includes many tests and supporting procedures indicating the results of each of the tests. The audit routines making up Version 3.0 of CCVS74 collectively contain all of the features in the High level of Federal Standard COBOL (FIPS PUB 21-1). The Communication Module has been excluded from this release of CCVS74.

1.1 BACKGROUND

The Federal COBOL Compiler Testing Service (FCCTS) is an activity of the Software Development Division of the Department of the Navy, Naval Data Automation Command, Federal Operations Directorate.

After July 1, 1972, all COBOL compilers brought into the Federal Government had to be identified as implementing one of the levels of the Federal COBOL Standard as defined in FIPS PUB 21. The National Bureau of Standards (NBS), which has the responsibility for the development and maintenance of Federal ADP Standards, delegated to the Department of Defense (DOD) the responsibility for the operation of a Government-wide COBOL Compiler Testing Service. This responsibility was discharged by the FCCTS through the implementation and maintenance of the 1968 COBOL Compiler Validation System (CCVS68), a comprehensive set of computer programs used to test COBOL compilers for compliance with the Federal COBOL Standard.

In anticipation of the updating of Federal Standard COBOL, the Software Development Division began producing a new COBOL Compiler Validation System which was based on the then proposed American National Standard Programming Language COBOL, X3.23-1974. Federal Standard COBOL was formally revised with the acceptance of FIPS PUB 21-1 by the Department of Commerce in August 1975. FIPS PUB 21-1 was formally published in December 1975, and all COBOL compilers brought into the Federal Government after June 1977 have to be identified as implementing one of the levels as defined in the revised Federal COBOL Standard.

The validation of COBOL compilers by the Federal COBOL Compiler Testing Service is done in support of the General Services Administration (GSA) Federal Property Management Regulation (FPMR) 101-32.1305-1a "Validation of COBOL Compilers". All COBOL compilers brought into the Federal inventory shall be validated. The test results for a COBOL compiler shall be used by a Federal Agency to confirm that a compiler meets the specifications of a designated level of Federal Standard COBOL (FIPS PUB 21-1), insofar as the COBOL Compiler Validation System tests the language elements included in that level of the Standard.

The development of the 1974 COBOL Compiler Validation System began in June 1973 and was based on the already existing 1968 CCVS. The programs making up the 1968 CCVS were converted in order to conform to the 1974 COBOL specifications. Additional programs were designed to test the new functions of existing modules as well as the new modules introduced into COBOL 74.

Version 1.0 of the 1974 CCVS was released in November 1975 and contained tests for the Low-Intermediate level of Federal Standard COBOL. Programs testing the High Federal level were added to the 1974 CCVS and released as version 2.0 in February 1977. With the formal announcement in the Federal Register in April 1977, Version 2.0 became the official set of audit routines for validating a COBOL Compiler's adherence to FIPS PUB 21-1. The official 1974 CCVS included programs for testing the Report Writer module but the Communication module was not included.

Version 3.0 of the 1974 CCVS contains programs which test the flagging requirement of FIPS PUB 21-1. Additional tests have also been included for language elements in the Nucleus, Table Handling, Inter-Program Communication and Indexed I-O modules. The Communication module is excluded from Version 3.0 of the 1974 CCVS.

1.2 PURPOSE AND NATURE OF COBOL COMPILER VALIDATION

The validation of a compiler, or any piece of software, determines the degree to which that product conforms to the technical specifications on which it was based. The use of compilers that have attained a high degree of conformance with their respective language standards (technical specifications) enhances source program interchangeability within all ADP installations which use that particular programming language.

The results of running a Compiler Validation System does not suggest the degree to which the compiler is usable (i.e., capable of data processing applications), but the degree to which individual language elements are usable. This gives an indication of conversion areas which must be considered in order to implement a source program from another computer system supporting a compiler based on the same set of language specifications.

Thus, the COBOL Compiler Validation System (CCVS) can be used to test a COBOL compiler's adherence to the standard language syntax, and, where unambiguous, language semantics of the technical specifications upon which the compiler is based. The latter, of course, is a more difficult area because of the lack of appropriate mechanisms for precise semantic specifications. The Validation System does not evaluate the implementation of a compiler nor its quantitative performance characteristics.

1.3 THE 1974 COBOL COMPILER VALIDATION SYSTEM

The 1974 COBOL Compiler Validation System (CCVS74) consists of audit routines produced in accordance with the technical specifications contained in COBOL 74 (X3.23-1974), their related data, and an executive routine (PPLVP74) which prepares the audit routines for compilation. Each audit routine is a COBOL source program which includes many tests and supporting procedures indicating the result of each test. The audit routines, from a design standpoint, collectively contain the features of American National Standard Programming Language COBOL, X3.23-1974, except for the Communications module, the ENTER

statement and arithmetic expressions in the Nucleus module.

The audit routines are maintained in a system independent source program library on magnetic tape. The use of magnetic tape is strictly for purposes of transportability. The source program library can be placed in a sequential mass-storage file, and in many instances, this is done during the validation of a compiler for purposes of convenience.

The executive routine extracts source programs from the source library and creates a file containing each audit routine with appropriate implementor names inserted in the source code. Additionally, the operating system control language statements required for compiling and executing each routine are generated before and after each source program as required. This file is then input to the operating system as a series of independent jobs.

The testing of a compiler in a particular hardware/operating system environment is accomplished by compiling and executing each audit routine. An output report produced by the execution of each audit routine indicates whether the compiler passed or failed each of the tests contained in the audit routine.

If the compiler rejects a language element by terminating compilation, issuing fatal diagnostic messages, or terminating execution abnormally, then the test containing the code the compiler was unable to process is deleted. After the offensive code is deleted, the audit routine is recompiled. A test is deleted by changing the source code in the test paragraph to comment statements, thereby allowing the compiler to produce a 'clean' object program.

The compiler listing (including any error messages), the output reports of each of the audit routines, and the updates shown in the output of the PPLVP74 processor constitute the raw data from which the members of the Federal COBOL Compiler Testing Service produce a Validation Summary Report (VSR). The VSR is the vehicle by which the Federal Government, for contracting purposes, determines the degree to which a COBOL compiler conforms to the requesting agency's requirements. The VSR is primarily intended to be used as part of the procurement process, but a VSR must be requested each time a COBOL compiler in the Federal Inventory is updated or changed.

1.4 OBTAINING VALIDATION SERVICES

1.4.1 Introduction

The NBS-DOD agreement (mentioned in 1.1 above) covers cost-reimbursable validations requested by:

- vendors wishing to have a compiler validated for their own purposes;
- vendors wishing to have a compiler validated in response to a Government request for proposals;
- Government agencies involved in a procurement;
- Government agencies wishing to validate a compiler already in use; or

- Other organizations, where the validation of a compiler benefits the Federal Government.

The results produced during an official validation are reviewed by the Federal COBOL Compiler Testing Service, which prepares a Validation Summary Report (VSR). The initial dissemination is to the requestor and the vendor who supports the compiler. If a requested validation has previously been performed on a similar computer configuration, the validation may not be repeated, in which case the earlier VSR is provided to the requestor. The VSR classifies a compiler according to each level of the Federal COBOL Standard for which support is claimed.

1.4.2 Requesting Validation Services

A REQUEST FOR VALIDATION SERVICES form may be obtained by writing to:

Director
Federal COBOL Compiler Testing Service (FCCTS)
Department of the Navy
Washington, D. C. 20376

The necessary information required to request a validation by the FCCTS and a sample validation request form is contained in Appendix F of this document.

1.5 OVERVIEW OF THE CCVS74 USERS GUIDE

The Users Guide explains the implementation and use of CCVS74 and the general guidelines used by the Federal COBOL Compiler Testing Service in conducting an official validation.

Section 2 goes into the organization and philosophy of CCVS74. It also explains the logical steps in implementing the system in general terms.

Section 3 identifies all of the software (source programs) comprising CCVS74. This includes all of the audit routines and two executive routines. The two executive routines (one is larger and more sophisticated than the second) are used to extract the source programs from the Population File (source program library) and prepare them for compilation on a particular system. During an official validation, one of these two executive routines is used to select the source programs to be run and for making changes to any of the source programs. The printed output from the executive routine is considered part of the raw data for producing the Validation Summary Report.

Section 4 of this document contains information necessary for ordering the complete detailed test specifications for the audit routines making up CCVS74.

Section 5 contains the information necessary to produce the environmental input for the CCVS74 executive routines. This information is used by the executive routine (VP-routine) to resolve all of the implementor-defined names which are used in the audit routines as the source programs are extracted from the Population File for compilation.

Section 6 describes the source language elements which can automatically be eliminated when the source programs are selected and prepared for compilation. These include language elements which do not need to be implemented to

meet the required language specifications in the COBOL Standard (e.g., CLOSE ...UNIT, VALUE OF phrase of the File Description entry). The automatic elimination of these language elements is in lieu of preparing source program updates to accomplish the same thing. This technique provides for consistency in running the audit routines when certain source code must be eliminated.

Section 7 is a 'checklist' which can be used in implementing CCVS74 and generating all the necessary input required by the executive routine to produce a source program for the system being validated. It gives the user a feel for what constitutes valid modifications to the source programs making up the audit routines in CCVS74.

Section 8 is made up of the Software Development Division's guidelines which are followed by the members of the Federal COBOL Compiler Testing Service when performing an official validation. These are provided to give the user an idea of what to expect in the way of required outputs and the methods used for validating a compiler in the event that an official validation is required.

Section 9 is a list of programs which must be compiled and executed in order to validate each of the four Federal Levels defined in FIPS PUB 21-1.

Section 10 describes and identifies the Population file (source program Library) for CCVS74 3.0.

Appendix A is the users guide for the executive routine (PPLVP74) which must be used in performing an official validation. This appendix contains all of the commands recognized by PPLVP74, the information necessary to describe the Job Control Language which should be generated for each source program selected from the Population File (source program Library), and the techniques by which updates can be made to the souce programs as they are selected.

Appendix B contains the information necessary to extract and implement the executive routine in support of CCVS74. This is made up of a sample COBOL source program (boot program) which can be used to extract the executive routine for compilation. It also contains the information necessary to use the executive routine to extract itself and the information for generating a new executive routine which differs from those extracted by the boot program.

Appendix C contains the messages and error diagnostics produced by the executive routine (PPLVP74).

Appendix D contains a brief description of the functions tested by each of the audit routines. It also contains the necessary environmental information which must be provided when each audit routine is selected.

Appendix E contains system dependent information for the implementation of the PPLVP74 processor. Examples are included for implementation under EXEC-8 (UNIVAC), GCOS (HIS) and OS/VS (IBM).

Appendix F contains information regarding the Federal COBOL Compiler Testing Service, FIPS PUB 21-1 and the information necessary to request validation services.

Appendix G contains Temporary Program Fixes (TPF's) which were discovered after the population file for this version of CCVS74 was produced. A list of the latest TPF's for any validation produced by the FCCTS is available. Requests should be forwarded to:

Director
Federal COBOL Compiler Testing Service
Department of the Navy
Washington, D.C. 20376

2. ORGANIZATION AND PHILOSOPHY OF THE COBOL COMPILER VALIDATION SYSTEM (CCVS)

2.1 PHILOSOPHY OF THE CCVS

The programs comprising the CCVS are maintained in a system independent source program library on magnetic tape. The programs are designed in such a way that all implementor names are presented in encoded mnemonics. These mnemonics are each made up of a series of X's followed by a three digit integer. The executive routine provided with the CCVS replaces these mnemonics with information provided by the user at the time the source programs are extracted from the source library. This results in syntactically correct programs for a given compiler. The techniques for identifying the mnemonics and providing the information necessary to resolve them are discussed in detail in Section 5 of this document.

There are certain language elements defined in the COBOL Standard which need not be supported in order for an implementation to conform to the Standard. The testing of these elements is accomplished through "optional code". The optional code can be selected as part of the source program or eliminated altogether through the use of an *OPTION command provided to the PPLVP74 processor. The various options are further described in Section 6 of this document.

Once the audit routines have been processed by PPLVP74, then, assuming the user-provided information was correct, the results are syntactically correct programs for the compiler being validated. The rejection of any of the source code by the compiler is a deviation from the technical specifications from which the CCVS is produced. It should be noted that this is by design, and no code which would be rejected by a compiler has been introduced in any of the source programs making up the CCVS.

2.2 ORGANIZATION OF THE CCVS

2.2.1 Introduction.

The Population File tape is a system independent source program library. It contains the executive routine (PPLVP74), environmental information, the audit programs, and all necessary data comprising the 1974 CCVS. Each program contains entries and statements that thoroughly exercise features of a portion of the language. Unless it is otherwise stated or implicit in the type of features being tested, each program is restricted to testing the features of a specific level of one functional module of COBOL. Some basic features of COBOL (such as MOVE, GO TO, PICTURE, IF, and SELECT) are used as supporting code in all of the audit routines. If any of these features functions incorrectly, the compiler may be dismissed immediately.

2.2.2 American National Standard Programming Language COBOL, X3.23-1974.

The specifications for ANS COBOL are based on a functional process concept. The language is divided into a Nucleus (internal processing) and eleven functional processing modules. The functional processing modules are Table Handling, Sequential I-O, Relative I-O, Indexed I-O, Sort-Merge, Report Writer, Segmentation, Library, Debug, Inter-program Communication, and Communication.

In addition, each module is further divided into two (low and high) or three (null, low, and high) levels.

In general, the sophistication and/or capability of the functional module increases with each higher level. Each level contains the capabilities of all lower levels within the same module. For example, level 2 Table Handling contains all the capabilities of level 1 Table Handling plus those features associated with level 2 Table Handling.

2.2.3 Federal Standard COBOL - FIPS PUB 21-1.

The specifications for Federal Standard COBOL are the language specifications given in X3.23.1974 minus the Report Writer module. The functional processing modules defined in X3.23-1974 are combined into four levels for Federal Standard COBOL. These four levels are identified as Low, Low-Intermediate, High-Intermediate and High. The X3.23-1974 modules and the module level contained in each Federal level are given in the following chart.

	Low Level	Low Intermediate Level	High Intermediate Level	High Level
NUCLEUS	1	1	2	2
FPMs				
TABLE HANDLING	1	1	2	2
SEQUENTIAL I-O	1	1	2	2
RELATIVE I-O	-	1	2	2
INDEXED I-O	-	-	-	2
SORT-MERGE	-	-	1	2
REPORT WRITER	-	-	-	-
SEGMENTATION	-	1	1	2
LIBRARY	-	1	1	2
DEBUG	-	1	2	2
INTER-PROGRAM COMMUNICATION	-	1	2	2
COMMUNICATION	-	-	2	2

2.2.4 Naming Convention for Audit Routines.

Each individual audit routine is named to associate it with the module and level of ANS COBL tested therein. Each module and level is represented by several programs. An audit routine name consists of five characters. The first two characters are alphabetic and identify the functional module being tested.

CM Communication
DB Debug
IC Inter-program Communication
IX Indexed I-O
LB Library

NC Nucleus
RL Relative I-O
RW Report Writer
SG Segmentation
SQ Sequential I-O
ST Sort-Merge
TH Table Handling

For the audit routines which are not flagging routines, the next character is either a 1 or a 2 and identifies the level of the functional module module.) The last two characters are 01, 02, 03, etc., to give unique names to each program containing features tested within the same level of the same functional module. For example, TH204 is the 4th in a series of several programs which test level 2 of the Table Handling module.

The five character program name for each flagging audit routine has the format XXYZN where:

XX is the 2-character name identifying the COBOL module.
Y is the integer 4 to identify the program as a flagging audit routine.
Z is the lowest Federal COBOL level in which this program is contained with

2 = Low-intermediate,
3 = High-intermediate,
4 = High.

N is a sequence number.

For example, RL421 represents a Relative I-O flagging audit routine which contains language elements found in the Low-Intermediate level of Federal Standard COBOL.

2.2.5 Choice of Audit Routines.

Because of the attributes described above, it is possible to test any combination of levels and modules of Federal Standard COBOL. The routines can either be selected individually by levels within modules, or in convenient groups ("subsets"). (The source program selection process including a description of the required PPLVP74 control statements to perform the selection is described in Appendix A.)

For example, in order to test level 1 of the Nucleus, level 2 of Table Handling, and level 2 of Relative I-O, it is necessary to select all programs which begin with the first three characters "NC1", "TH1", "TH2", "RL1", and "RL2". (Note that, according to ANS specifications, the higher level(s) of a given module technically consist of the lower level plus some unique "higher-level features". Therefore, in order to completely test any higher-level module, it is necessary to select all lower-level tests for the module as well.)

The first two characters of a flagging program-name identify the ANS module, but the ANS module level is not part of the routine name. Instead, the Federal level is indicated by the name. In order to completely test a particular ANS module, it is necessary to correlate the ANS module level to the Federal

level in which it appears. As an example, to completely test level 2 of the Relative I-O module, which is contained in High-Intermediate Federal Standard COBOL, the flagging routines RL421 and RL431 must also be selected.

2.3 STEPS FOR USING THE 1974 COBOL COMPILER VALIDATION SYSTEM.

2.3.1 General.

Five distinct steps are to be taken in using the audit routines. In some steps manual operations are required, and in two of the steps it will frequently be necessary to go back to previous steps.

Many users will find the entire set of audit routines too voluminous to introduce into the system in one pass. If this is the case, the audit routines can be divided into several portions in any way the user wishes, and each portion can be run separately, beginning with step two. Step one needs to be performed only once, regardless of the apportionment. The entire process is described in paragraphs 2.3.2 through 2.3.6.

2.3.2 Step One - PPLVP74 Processor (Executive Routine).

The PPLVP74 processor which is written in COBOL must be extracted from the Population File. This can be done using the sample program described under PPLVP74 GENERATION in Appendix B. When modified appropriately, the sample program will extract the PPLVP74 processor, place the correct implementor-names in the appropiate places, establish table sizes for the various control information used by the processor, and select all of the necessary optional source statements to produce a complete program. The PPLVP74 processor is then compiled for subsequent use in the manipulation of the Population File (source program library) and its contents.

2.3.3 Step Two - PPLVP74 Input Parameters and Data.

Before the PPLVP74 processor (or executive routine) can be used, input data must be prepared for it. This input includes the information necessary to (1) select the source programs to be extracted, compiled and/or executed; (2) resolve all of the implementor-names which may be present in the Environment and Data Divisions (see Section 5 for a list of the mnemonics); (3) establish which options are to be used in regard to optional source cards (see Section 6); (4) describe a skeleton of the Job Control Language to be generated for each program or set of programs (see Appendix A).

2.3.4 Step Three - PPLVP74 Execution.

Step three consists of executing the PPLVP74 processor. There are two primary output files: PRINT-FILE and SOURCE-COBOL-PROGRAMS. SOURCE-COBOL-PROGRAMS, or source file, consists of audit routines in the form of COBOL programs, copied with a small number of changes from the Population File. These changes include the insertion of meaningful implementor-names and job control language. PRINT-FILE, the PPLVP74 listing, contains a summary of these changes plus error messages generated by the processor, if appropriate, plus some minor accounting information.

The user can determine from the listing whether the source file has been

created satisfactorily. Each possible message on this listing is explained in Appendix C. If the PPLVP74 processor listing indicates that the source file is not in the form desired by the user, it can be corrected by returning to step two and modifying the PPLVP74 control cards, and repeating step three. When a satisfactory source file is created, the validation continues with step four.

2.3.5 Step Four - Introducing the Audit Routines into the Operating System Environment.

Step four introduces the output from the PPLVP74 processor into the operating system as input. Each of the programs selected should be ready to execute as a series of jobs from the source file. The method used to enter this jobstream into the system is implementor defined. All of these factors should have been considered in preparing the PPLVP74 processor control cards in step two, and when describing the source file to be produced by the PPLVP74 processor in step one.

If the user exercises his option to build the COBOL copy library for use in validating the Library module and the computer has multiprogramming capabilities, the user should include instructions in the jobstream to assure that the copy library is completely built before the audit routines that interact with the Library begin to compile. If this is not possible, the copy library file should be built in an earlier independent run.

The audit routines are designed to run with a minimum of operator intervention. All files used are created by the audit routines, immediately used, and scratched. No more than five tape (sequential) files and three mass-storage files are used by any single program. Only one, or at most two, audit routines require direct interaction with the operator. Pertinent details about every audit routine are discussed in Appendix D. During an official validation, one set of Sequential I-O programs requires a tape input file created on another computer system to determine whether the compiler and operating system support ASCII and related data interchange standards.

2.3.6 Step Five - Examining the Results of Compiling/Executing the Audit Routines.

The last step is to examine the results of the compilation and execution of each audit routine. If there are any errors in the compilation of an audit routine, the execution report must be considered invalid. The user is not to dismiss as "unimportant" any errors that the compiler designates as "fatal". When any fatal compiler error occurs, the user must make update cards as described in Appendix A.4 to eliminate the errors. These updates are added to the to the PPLVP74 control card deck, and steps 3, 4, and 5 are repeated until a valid report is generated.

The results of the execution of each audit routine are presented in the form of a printed report. Each test in the audit routine is described by the words "PASS", "FAIL",, or "TEST DELETED", along with a brief description of the nature of the test and the name of the COBOL program paragraph that contains the source code making up the test. Each "FAIL" on the report is usually accompanied by further information to help determine the cause of the failure.

The raw data collection for a validation is complete when a valid report is achieved for each of the audit routines being used.

2.4 DATA FOR AUDIT ROUTINES.

2.4.1 General.

For the most part, the necessary data for the programs are created either internally or by the previous program within that module and level. The library module requires external data to produce a system copy library complete with the text to be copied into source programs. This text is provided on the Population File. Proper PPLVP74 control cards will create a job that will place the data on a file in accordance with the implementor's library format to be used by the Library module tests. The two Nucleus programs testing the ACCEPT verb also use external data, located on the Population File immediately following each program. If the system allows ACCEPTing from the input device that contains the source program, the PPLVP74 processor can automatically place the necessary system control language in front of and behind the data. All techniques are explained in Appendix A (A.3).

2.4.2 Data-Passing Sets.

A method frequently used to generate data is to designate two, and sometimes three, consecutive programs as a data-passing set of programs. The method of operation is for the main program to generate output data which is saved by the system to be used as input to the next program. The second program may, in turn, generate output which becomes input to a third program. In general, the final program of a set generates the most significant report, while the earlier programs in the set generate either a token report or no report at all. Regardless of the content, no single program of a data-passing set may be run without running, in proper order, all of the other programs of that set.

2.5 PROGRAM DOCUMENTATION.

Documentation for each audit routine is contained in the comment entries of the IDENTIFICATION DIVISION. Additional information about each specific test is contained in comment lines within the PROCEDURE DIVISION. The formal documentation and detailed test specifications for the CCVS is available from the National Technical Information Service. (See Section 4 of this document.)

2.6 AUTOMATION OF THE AUDIT ROUTINES (PPLVP74).

2.6.1 General.

The PPLVP74 processor is used in order to automate the handling of the audit routines. Its function is to create an output file (file name SOURCE-COBOL-PROGRAMS also referred to as "Source File" in this documentation), which is in a form acceptable to the user's computer as a system input jobstream. Below are two possibilities which may be used:

a. Create an updated system COBOL library file which allows each of the programs to be compiled and executed by means of the system external control. Updating or altering of the source COBOL programs may be done either by the system's updating utilities or

by the VP-Routine.

b. Create a system input file with all the necessary operating system control language to automatically compile and execute the audit routines. Updating and changing of the programs must then be accomplished through the use of PPLVP74.

Due to the differences among different compilers and operating systems, the audit routines must be modified to operate on each system. In addition to the necessary changes in the ENVIRONMENT DIVISION, the format of the modified audit routines must conform to the system requirements for COBOL library files or system input files (jobstream files), whichever is used. The file of modified audit routines, called the Source File, is built by reading the necessary control cards and environment information as data into the PPLVP74 processor, which in turn merges this information with the information contained on the Population File.

To facilitate the use of the Population File tape and allow for the various input/output requirements among systems, the PPLVP74 is designed to do its own blocking and deblocking. The tape as received will consist of thirty 80-character records per block or 2400 characters per record. The contents of the tape file can be placed on a mass-storage device as a sequentially organized file if desired.

2.6.2 Monitor and Update Sections.

The processing done by the PPLVP74 can be divided into two distinct stages. These two stages correspond to, and are controlled by, the two groups of cards which comprise the card input to the PPLVP74 processor. The first group of cards comprise the Monitor Section control cards and the second group are the Update Section control cards. The two stages are called the Monitor Stage and the Source File Creation Stage, respectively.

2.6.3 General Format of the PPLVP74 Control Deck.

Three control cards must be present, regardless of the other cards in the deck. Their formats are given below, in the order they are to appear in the deck:

```
0         1         2
12345678901234567890

(Monitor Control Information)
*END-MONITOR
*BEGIN-UPDATE
(Update Control Information)
*END-UPDATE
```

2.6.4 Monitor Section Control Cards.

Monitor Section control cards describe the parameters that will be used during the Source File Creation Stage. During the Monitor Stage, the PPLVP74 reads these cards one at a time, sets logical switches as a result of a few of them, and builds tables with information contained on the others. All Monitor Section cards must precede the *END-MONITOR card and any updates to the pro-

grams must follow the *BEGIN-UPDATE card (in order of appearance in the population file) and before the *END-UPDATE card.

2.6.5 Update Section Control Cards.

The Update Section control cards are read during the Source File Creation Stage. They are used to modify the audit routines as they are being written on the Source File, and to effect temporary modifications to the parameters described by the Monitor Section cards. All Update Section cards must be placed between the *BEGIN-UPDATE and *END-UPDATE cards.

No PPLVP74 control statements are ever placed between the *END-MONITOR and *BEGIN-UPDATE cards or after the *END-UPDATE card.

2.7 VP-ROUTINE FILES

2.7.1 SOURCE-COBOL-PROGRAMS (Source File)

The SOURCE-COBOL-PROGRAMS file is an output file that contains the selected programs which have been updated, the assignment replacements, and the job control language. It has the following chacteristics:

- o Record: 80 or 72-character image.
- o Blocking: Unblocked.
- o Label: Omitted (unlabeled).
- o File Mark: After last program of the file.

This file may be blocked by so specifying in the File Section of the PPLVP74 source program. It is permissible for this file to reside on a device other than tape.

2.7.2 UPDATED-POPULATION-FILE

UPDATED-POPULATON-FILE is an output file that is used during maintenance to create a new version of the CCVS. Its format is the same as that of POPULATION-FILE, given below. During a validation, it is to be regarded as an unused scratch file.

2.7.3 PRINT-FILE

PRINT-FILE is an output file that gives a listing of the control cards and the selected/updated programs. The amount of information contained in the PRINT-FILE can be controlled by the *LIST card (paragraph A.2.13).

2.7.4 POPULATION-FILE

POPULATION-FILE is an input file that contains all of the programs in the source program library. The file has the following characteristics:

- o Record: Thirty 80-character records per block giving a 2400 character record.
- o Blocking: Deblocking is done by the program.
- o Label: Unlabeled (omitted).
- o File Mark: After last program on the file.

It is permissible for this file to reside on a device other than tape.

2.7.5 CONTROL-CARD-FILE (PPLVP74 control input).

CONTROL-CARD-FILE is an input file used to supply user information and instructions to the PPLVP74 processor, plus any updates to the Population File which might be needed. The file is unlabeled and described as 80 characters per record. This file can be asigned to whatever input device is appropriate.

3. CCVS74 3.0 SOURCE PROGRAMS

Version 3.0 does not contain all of the programs that will finally comprise the 1974 CCVS. This is due to two reasons.

(1) The existing programs will be enhanced and additional programs produced as more experience is gained and more COBOL 74 compilers become available.

(2) There are some areas of the COBOL 74 technical specifications which require interpretation before any meaningful testing can be accomplished.

The following list shows, by module, the programs and other system software provided in Version 3.0 of CCVS74. The name given is referenced as described under program selection in the documentation for the PPLVP74 processor. (Parenthesized names need not be selected. These programs are automatically selected when the preceeding "main" program is selected. These programs involve the passing of data files and/or CALLing programs.) The individually named programs can be selected for processing by the PPLVP74 processor through the use of a "+" card (e.g., +DB101) or an entire level or module can be selected through the use of an *ENVIRONMENT card (e.g., *ENVIRONMENT DB100 or *ENVIRONMENT DB100, DB200, IC100...). See Appendix A.2, Source Program Selection (Monitor) Section) for more information.

The programs whose names contain the number "4" as the third character are designed to be used to determine the degree to which a COBOL Compiler meets the flagging requirements defined in FIPS PUB 21-1. These programs have a dual function in that they also contain tests which make up the COBOL Compiler Validation System.

3.1 System Software (Not part of CCVS74).

```
ASVP0+    PPLVP74 (HYP0)
ASVP9     PPLVP74
```

3.2 Communication Module

```
Withheld from CCVS74 3.0
CM431 (for flagging purposes only)
```

3.3 Debug Module

```
DB101 through DB105     (*ENVIRONMENT  DB100)
DB421 through DB422
DB201 through DB204     (*ENVIRONMENT  DB200)
DB431
```

3.4 Inter-program Communication Module

```
IC101                   (*ENVIRONMENT IC100)
    (IC102)
IC103
    (IC104)
    (IC105)
```

```
      IC106
          (IC107)
      IC108
          (IC109)
          (IC110)
          (IC111)
      IC112
          (IC113)
      IC114
          (IC115)
      IC116
          (IC117)
          (IC118)
      IC119
          (IC120)
      IC121
          (IC122)
          (IC123)
      IC421
          (IC422)

      IC151+                    (*ENVIRONMENT IC000)
          (IC152)+

      IC201                     (*ENVIRONMENT IC200)
          (IC202)
      IC203
          (IC204)
          (IC205)
          (IC206)
      IC207
          (IC208)
      IC431
          (IC432)
```

3.5 Indexed I-O Module

```
      IX101                     (*ENVIRONMENT IX100)
          (IX102)
          (IX103)
      IX104
      IX105
          (IX106)
      IX107

      IX201                     (*ENVIRONMENT IX200)
          (IX202)
          (IX203)
      IX204
      IX205
      IX206
      IX207
      IX208
      IX209
      IX210
```

IX211
IX441
IX442

3.6 Library Text

ALTL1 (*ENVIRONMENT K0000)
ALTLB
KP001
KP002
KP003
KP004
KP005
KP006
KP007
KP008
K1FDA
K1PRA
K1PRB
K1P01
K1SEA
K1WKA
K1WKB
K1WKY
K1WKZ
K1W01
K1W02
K1W03
K1W04
K101A
K2PRA
K2SEA
K3FCA
K3FCB
K3IOA
K3IOB
K30CA
K3SCA
K3SNA
K3SNB
K4NTA
K5SDA
K5SDB
K501A
K501B
K6SCA
K7SEA

3.7 Library Module

LB101 (*ENVIRONMENT LB100)
(LB102)
LB103
(LB104)
LB105

```
     LB106
     LB107
     LB421

     LB201                    (*ENVIRONMENT LB200)
         (LB202)
     LB203
         (LB204)
     LB205
     LB206
     LB207
     LB441
```

3.8 Nucleus Module

```
     NC101 through NC120      (*ENVIRONMENT NC100)
     NC151 through NC165+     (*ENVIRONMENT NC000)
     NC201 through NC219      (*ENVIRONMENT NC200)
     NC431
```

3.9 Relative I-O Module

```
     RL101                    (*ENVIRONMENT RL100)
         (RL102)
         (RL103)
     RL104
     RL105
     RL106
     RL107
     RL108
         (RL109)
     RL421

     RL151+                   (*ENVIRONMENT RL000)
         (RL152)+
         (RL153)+

     RL201                    (*ENVIRONMENT RL200)
         (RL202)
         (RL203)
     RL204
     RL205
     RL431
```

3.10 Report Writer Module

```
     RW101
     RW102
     RW103
     RW104
```

3.11 Segmentation Module

```
     SG101 through SG106      (*ENVIRONMENT SG100)
     SG421
```

```
      SG201 through SG204     (*ENVIRONMENT SG200)
      SG441
```

3.12 Sequential I-O Module

```
      SQ101 through SQ111     (*ENVIRONMENT SQ100)
      SQ112
          (SQ113)
      SQ114 through SQ117
      SQ118
          (SQ119)
          (SQ120)
      SQ121

      SQ151 through SQ153+    (*ENVIRONMENT SQ000)

      SQ201                   (*ENVIRONMENT SQ200)
      SQ202
      SQ203
          (SQ204)
      SQ205
      SQ206
      SQ207
          (SQ208)
          (SQ209)
      SQ210
      SQ211
      SQ212
      SQ213
      SQ214
      SQ215
      SQ216
      SQ217
      SQ218
      SQ219
      SQ220
      SQ431
```

3.13 Sort-Merge Module

```
      ST101                   (*ENVIRONMENT ST100)
          (ST102)
          (ST103)
      ST104
          (ST105)
      ST106
          (ST107)
      ST108
      ST109
          (ST110)
          (ST111)
      ST112
          (ST113)
          (ST114)
      ST115
```

```
    (ST116)
    (ST117)
ST118
ST431
    (ST432)
    (ST433)
    (ST434)

ST201 through ST216     (*ENVIRONMENT ST200)
ST441
    (ST442)
    (ST443)
```

3.14 Table Handling Module

```
TH101 through TH111     (*ENVIRONMENT TH100)
TH151 through TH152+    (*ENVIRONMENT TH000)
TH201 through TH221     (*ENVIRONMENT TH200)
TH431
```

3.15 Environments for Flagging Routines

There are four environments defined for the flagging routines. One environment contains all of the flagging routines and three environments group the routines according to their Federal level. The environment identification and the routines each contain are:

ENVIRONMENT	CONTENTS
FL400	All flagging routines.
FL420	Low-Intermediate level routines. (DB421, IC421, IC422, LB421, RL421, SG421)
FL430	High-Intermediate level routines. (CM431, DB431, IC431, IC432, NC431, RL431, SQ431, ST431, ST432, ST433, ST434, SQ431, TH431)
FL440	High level routines. (IX441, SG441, ST441, ST442, ST443)

+These programs collectively represent the programs necessary to validate the U.S. Navy HYPO-COBOL for minicomputers. This subset of COBOL 74 was produced by the U. S. Navy in the absence of any other available specifications produced specifically to permit a small subset of COBOL to be used on a minicomputer and at the same time take advantage of the technology exhibited by the average general purpose minicomputer. The HYPO-COBOL technical language specifications, and the detailed test specifications can be obtained from the National Technical Information Service. See Section 4 of the Users Guide.

4. CCVS74 DOCUMENTATION

Documentation and detailed test specifications for the full 1974 CCVS has been available from the National Technical Information Service (NTIS) since December 1974. A complete list of the Test Specifications and the other CCVS74 products follows. The leftmost codes, in each case beginning with the letters "ADA", are the NTIS accession (order) codes.

1974 CCVS TEST SPECIFICATIONS

ADA002792	USN-CCVS-74-VOL-01	NUCLEUS Level 1	$15.25
ADA006921	USN-CCVS-74-VOL-02	NUCLEUS Level 2	$12.50
ADA002084	USN-CCVS-74-VOL-03	TABLE HANDLING	$ 7.50
ADA019757	USN-CCVS-74-VOL-04	SEQUENTIAL I-O	$ 3.75
ADA003072	USN-CCVS-74-VOL-05	RELATIVE I-O	$ 3.75
ADA007554	USN-CCVS-74-VOL-06	INDEXED I-O	$ 5.25
ADA002384	USN-CCVS-74-VOL-07	SORT-MERGE	$ 3.75
ADA002385	USN-CCVS-74-VOL-09	SEGMENTATION	$ 4.25
ADA002386	USN-CCVS-74-VOL-10	LIBRARY	$ 4.25
ADA030461	USN-CCVS-74-VOL-11	DEBUG	$ 3.25
ADA002975	USN-CCVS-74-VOL-12	INTER-PROGRAM COMM.	$ 3.75
ADA002388	USN-CCVS-74-VOL-13	COMMUNICATION	$ 4.25

1974 CCVS VERSION 3.0(*)

ADA036174	CCVS74 V3.0 User's Guide	$ 12.00
ADA036173	CCVS74 V3.0 Population File (TAPE)	$500.00

(*)The V3.0 accession numbers were not available when this document was prepared. These are the V2.0 accession numbers and the replacement documents (V3.0) can be obtained by referring to these accession numbers.

HYPO-COBOL COMPILER VALIDATION SYSTEM

ADA018916	HCCVS Language Specifications	$ 6.25
ADA024915	HCCVS Test Specifications	$11.75
ADA024914	HCCVS Population File (TAPE)	$200.00

The mailing address for NTIS is:

National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22151

5. CCVS74 X-Cards

X-cards provide the ability to transport the audit routines to different computer system environments. The list below explains the function of every X-nn number substitution currently in use in the audit routines making up the 1974 COBOL Compiler Validation System (CCVS). A user running the complete set of audit routines must create an X-nn card to correspond to every active number in the list.

If there is a period (.) in the source program following the XXXXXnnn card specification, a period will follow the substituted X-card data in the source produced by the running of the VP-Routine.

```
SOURCE-COMPUTER.          In audit routine source;
    XXXXX082.             note period in original
                          source.

SOURCE-COMPUTER.          Final source after PPLVP74
    BRAND-X.              substitution; period follows
                          substituted source.

X-82    BRAND-X.          X-card specification for
                          this example; note period
                          follows the data to be
                          substituted.
```

If there is no period in the source program following the XXXXXnnn card specification, there will be no period following the substituted X-card data in the source produced by the execution of the PPLVP74 processor.

```
VALUE OF                  In audit routine source;
XXXXX074                  no period in original
IS                        source.

VALUE OF                  Final source after VP-
ID                        routine substitution; no
IS                        period in substituted source.

X-74    ID.               X-card specification for
                          this example; note period
                          follows the data to be
                          substituted.
```

The X-card (also called the "X-nn card") is used to enter the implementor or user-defined word or phrase into the ENVIRONMENT-TABLE. The X-card format is as follows:

```
0        1         2         3.............                ....8
12345678901234567890123456789O                               0
X-nnmm implementor or user-defined word, literal, or phrase.
```

Columns 1 and 2 contain "X-". The number "nn" is the same as its counterpart on the XXXXXnnn card. The "mm" represents the starting position in the replaced source image in which the source program will be placed. (See Appendix A. for further information regarding X-cards and the options available for them.)

The implementor or user-defined word or phrase may appear anywhere in columns 8 through 68. Each implementor-defined or user-defined word or phrase must be terminated with a period. Enclose all nonnumeric literals in quotes (") terminated by a period. During the replacement, column 8 becomes column 12 in the COBOL program generated by the PPLVP74 processor. Column 9 becomes 13, ... column 68 becomes 72. The PPLVP74 processor uses nn as an occurrence number when storing (and later retrieving) the implementor-defined word or phrase in the ENVIRONMENT-TABLE.

The user must remember to create corresponding A-nn (and possibly APnn and ADnn) cards where needed. This is only required for files which require job control language to be generated to assign, pass, and/or delete them. (This should only apply to X-cards X-01 thru X-58.) Some system files used in data passing sets of audit routines must be formally "purged" to avoid confusion when the same file name is used for subsequent routines.

In the source programs, X-cards for substitution are in the format XXXXXnnn. In the environment lists for A-cards and X-cards, the format is A-nn and X-nn respectively. The source card XXXXX001 corresponds to X-01, A-01 (and perhaps AP01 plus AD01). If for some reason a system requires a user to expand the X-card list beyond X-99, use the format XXXXXmmm for the X-card number mmm in the source program and a corresponding X-card of Xmmm in the list of environmental substitutions. For example, if the list needed to be expanded through the number 169, use XXXXX169 in the source program and X169 in the X-card list.

X-card	Function	Explanation
X-01	Tape name 1	Implementor-name in ASSIGN TO clause of SELECT statement; e.g., SELECT file-name ASSIGN TO XXXXX001. X-01 UNISERVO TAPE-1. A-01 @ASG,T TAPE-1.,8C/1
X-02	Tape name 2	
X-03	Tape name 3	
X-04	Tape name 4	
X-05	Tape name 5	
X-06	Multireel name 1	Implementor-name in ASSIGN TO clause, 3 reels.
X-07	Multireel name 2	Implementor-name in ASSIGN TO clause, 2 reels.
X-08	Multifile-reel name 1	
X-09	Multifile-reel name 2	
X-10	Multifile-reel name 3	Implementor-name in ASSIGN TO clause of SELECT statement; all files are assigned to the same physical reel of tape.
X-11	Multifile-reel name 4	
X-12	Multifile-reel name 5	
X-13	Multifile-reel name 6	
X-14	Sequential mass storage name 1	Implementor-name in ASSIGN TO clause of SELECT statement; organization is sequential.
X-15	Sequential mass storage name 2	
X-16	Sequential mass storage name 3	
X-17	Optional multireel tape name 3	Implementor name in ASSIGN TO clause of SELECT OPTIONAL statement for a three reel file; this file should not be present at execution time.
X-18	Optional sequential mass storage name 4	Implementor-name in ASSIGN TO clause of SELECT OPTIONAL statement; this file should not be present at execution time.
X-19	Multi-unit name 1	Implementor-name in ASSIGN TO clause of SELECT statement; 3 mass storage units each for X-19 and X-20; used with CLOSE UNIT statements.
X-20	Multi-unit name 2	
X-21	Relative file name 1	

X-22	Relative file name 2	Implementor-name in ASSIGN TO clause of SELECT statement; organization is relative.
X-23	Relative file name 3	
X-24	Indexed file name 1	Implementor-name in ASSIGN TO clause of SELECT statement; organization is indexed (See X-44, X-45 and X-46).
X-25	Indexed file name 2	
X-26	Indexed file name 3	
X-27	Sort file name 1	Implementor-name in ASSIGN TO clause of SELECT statement for sort files.
X-28	Sort file name 2	
X-29	Sort file name 3	
X-30 - X-43	NOT USED	
X-44	System-name for indexed file name 1	Implementor-name for those systems which require an additional system interface for the indices in an indexed file (*OPT10 J).
X-45	System-name for indexed file name 2	
X-46	System-name for indexed file name 3	
X-47	Copy library name 1.	
X-48	Copy library name 2.	Referenced in COPY statement for alternate library.
X-49	Sequential file for a Report File	(Report Writer)
X-50	Sequential file for a Report File	(Report Writer)
X-51	Switch name 1	Implementor-name in SPECIAL-NAMES statement; this switch should be ON at execution time.
X-52	Switch name 2	Implementor-name in SPECIAL-NAMES statement; this switch should be OFF at execution time.
X-53	RERUN statement	Complete entry in the I-O-CONTROL paragraph; e.g.,

RERUN ON SQ-FRR EVERY 10 RECORDS OF RR-FS1.

SQ-FRR is a file provided in each of the audit routines which use the rerun facility. It is defined but not referenced in the Procedure Division. It is strictly for use in the Rerun statement if a

file-name option is required by the implementation.

RR-FS1 is the file which is accessed in the Procedure Division. It is the same name in each program which audits the Rerun facility. Rerun information is written on SQ-FRR whenever 10 records of RR-FS1 have been processed.

X-54	System punch	Implementor-name in ASSIGN TO clause of SELECT statement.
X-55	System printer	Implementor-name for output listings in ASSIGN TO clause of SELECT statement.
X-56	DISPLAY mnemonic name	Implementor-name in SPECIAL-NAMES statement.
X-57	ACCEPT mnemonic name	Implementor-name in SPECIAL-NAMES statement.
X-58	System input device	Implementor-name in ASSIGN TO clause of SELECT statement.
X-59	Alphabet-name	Implementor-name to be used as an alphabet-name.
X-60	Not used	
X-61	Not used	
X-62	Not used	

X-63 A 51 character alphanumeric literal containing 50 of the characters of the COBOL character set. The characters are in ascending order according to the native collating sequence; the quotation mark (") is excluded and the currency symbol ($) is repeated twice. The literal is bounded by quotation marks and terminated by a period. For a system whose native collating sequence is ASCII, the following literal is used:

" $$()*+,-./0123456789;<=>ABCDEFGHIJKLMNOPQRSTUVWXYZ".

X-64 A 51 character alphanumeric literal made up of any 51 characters. The characters are in descending order according to the native collating sequence. The literal is bounded by quotation marks and terminated by a period.

X-65 Number of records generated and sorted in ST115-ST117. For an official validation this four digit integer must be 9000.

X-66 Not used

X-67 Memory size in words — Implementor supplied integer specified in the MEMORY SIZE

X-68	Memory size in characters	clause of the OBJECT-COMPUTER paragraph.
X-69	Additional VALUE OF phrase for each FD containing a VALUE OF phrase. This image is optionally generated; see *OPT8 card.	
X-70	Not used	
X-71	VALUE OF phrase 8	(See X-74.)
X-72	VALUE OF phrase 7	(See X-74.)
X-73	Implementor-name for top of page	Implementor-name defined in SPECIAL-NAMES paragraph, associated with mnemonic-name in WRITE ADVANCING statement.
X-74	VALUE OF Implementor-name	In VALUE OF clause for standard labels; e.g., VALUE OF XXXXX074 IS XXXXX075 (or XXXXX076, 077, 078, 079, 080, 071, 072)
X-75	VALUE OF phrase 1	(See X-74.)
X-76	VALUE OF phrase 2	(See X-74.)
X-77	VALUE OF phrase 3	(See X-74.)
X-78	VALUE OF phrase 4	(See X-74.)
X-79	VALUE OF phrase 5	(See X-74.)
X-80	VALUE OF phrase 6	(See X-74.)
X-81	An eight character alphanumeric literal containing eight unique characters which are contained in the native character set but are not contained in the 51 character COBOL character set. The literal is bounded by quotation marks and terminated by a period.	
X-82	Source computer name	Implementor-name defined in SOURCE-COMPUTER paragraph.
X-83	Object computer name	Implementor-name defined in OBJECT-COMPUTER paragraph.
X-84	LABEL records option for the printer destined file used by each of the audit routines. If not specified, OMITTED will be assumed. If the word STANDARD appears, additional information may also be required in this entry. (i.e., VALUE OF, CODE-SET IS ,...)	

The following example is provided in order to explain X-85 through X-88.

POPULATION FILE	RESOLVED SOURCE PROGRAM
FD file-name-1	FD file-name-1
VALUE OF	VALUE OF
XXXXX074	FILE-ID
IS data-name-1	IS data-name-1
XXXXX085	RETENTION-DATE
IS data-name-2.	IS data-name-2.
FD file-name-2	FD file-name-2
VALUE OF	VALUE OF
XXXXX074	FILE-ID
IS	IS
XXXXX075	"FILE 2"
XXXXX085	RETENTION-DATE
IS data-name-3.	IS data-name-3.
01 data-name-1	01 data-name-1
XXXXX086.	PIC X(6) VALUE "FILE 1".
01 data-name-2	01 data-name-2
XXXXX087.	PIC X(6) VALUE "770131".
01 data-name-3	01 data-name-3
XXXXX088.	PIC X(6) VALUE "770228".

X-85	VALUE OF Implementor-name	Second implementor-name in the VALUE OF clause in File Description entry for standard labels.
X-86	PICTURE description and VALUE clause	Data Description entry for data-name-1 associated with VALUE OF clause in File Description entry.
X-87	PICTURE description and VALUE clause	Data Description entry for data-name-2 associated with VALUE OF clause in File Description entry.
X-88	PICTURE description and VALUE clause	Data Description entry for data-name-3 associated with VALUE OF clause in File Description entry.

6. CCVS74 Option Switches

The concept of optional code within the programs making up the CCVS permits the elimination or inclusion of source code whose implementation is not required in order to conform to the COBOL specifications. This enables the use of programs containing these optional elements without having to make updates to the source program.

The functions/language elements which can be included or eliminated through the use of option switches include the VALUE OF clause associated with mass storage files, switch status tests, CLOSE REEL statements, CLOSE UNIT statements, and OPEN ... REVERSED statements. Additionally, the use of an option switch can cause coding to be generated to dump all files used in the I-O programs to the printer for examination of their contents, and control the type of WRITE statement used for printer destined file created by each audit routine. The format for the *OPTn control card is shown in A.2.16.

For a given switch there are two or more choices. If there is no indication that a switch can be blank, then blank constitutes an invalid switch setting. This is not detected by the PPLVP74 processor but will affect the source programs which are dependent on the switch for proper source code generation. The defaults (those switch settings which are assumed for use with CCVS74 by the PPLVP74 processor and need not be set) are indicated in the following table. If no deviations are necessary from the default switch settings then no *OPT cards need be present.

The following meanings are attached to each number/letter combination for the 1974 CCVS. Switches 1 through 4 are also relevant to the 1968 CCVS.

Switch Number	Value	Comment
1	A	Generate all source statements referring to system switches (Default).
	B	Delete all references to system switches.
2	E	Generate all CLOSE UNIT statements. (Default)
	F	Delete all CLOSE UNIT statements.
3	H	Generate all CLOSE REEL statements. (Default)
	I	Delete all CLOSE REEL statements.
4	L	Generate all references to 'REVERSED' in all OPEN statements (Default).
	M	Delete all references to 'REVERSED' in all OPEN statements.
6	X	Generate dump routine to dump all files in I-O

		audit routines in which one or more of the tests failed.
	blank	Eliminate the dump coding (Default).
7	Y	WRITE statement for the audit routine report will use the AFTER PAGE and AFTER integer options (Default).
	Z	WRITE statement for the audit routine report will use the AFTER integer option of the WRITE statement. The first record written will be 'AFTER ADVANCING 15 LINES' to provide spacing between the Audit Routine Report and whatever may preceed the report in the print queue.
	blank	WRITE statement for the audit routine report will use the AFTER integer option of the WRITE statement and the heading will appear as the first record after the file is opened.
8	C	Generate the 'VALUE OF ' phrase for all File Description entries (Default).
	blank	Suppress the 'VALUE OF ' phrase for all File Description entries.
9	G	Generate the optional X card for the additional phrases of the VALUE OF clause in the File Description entry. (X-69)
	blank	X-69 is suppressed (Default).
10	J	Generate the extra X-card for indexed I-O files.
	blank	Additional X-card is suppressed (Default).
11	T	Establish the RECORD KEY and ALTERNATE KEYs as being 29 characters for indexed files (Default).
	U	Establish the RECORD KEY and ALTERNATE KEYs as being no greater than 8 characters for indexed files.
25	W	FCCTS debug code - not for use during a validation.
	blank	FCCTS debug code will be suppressed (Default).

The following checklist can be used to code the options to be used during a validation:

```
*OPT1   -
*OPT2   -
*OPT3   -
*OPT4   -
*OPT6   -
*OPT7   -
*OPT8   -
*OPT9   -
*OPT10  -
*OPT11  -
```

7. CCVS74 IMPLEMENTATION TECHNIQUES

The implementation of CCVS74 is accomplished in several distinct phases. This discussion presumes that the executive routine provided will be used in implementing CCVS74. During an official validation, the executive routine must be used to select and prepare the source programs for compilation. The information necessary to implement the executive routine (PPLVP74) is contained in Appendix B.

7.1 SOURCE PROGRAM RESOLUTION

The first phase of the implementation of CCVS74 is the resolution of implementor names within the source program. Within the programs making up CCVS74, the implementor names have been defined to the extent that there is a symbol which represents each implementor name. As each of these symbols are defined, they are then used consistently throughout the rest of the audit routines. These symbols are made up of several Xs followed by a file disposition indicator, followed by a three digit integer. Symbols which interface the COBOL program SELECT statement with the operating system use the file disposition (an optional letter in place of the last X) to convey information to the PPLVP74 processor. This letter, if present, further defines the symbol and is described in the alphabet-cards section of Appendix A. The symbol begins in column 12 of the source image in which it is defined. The rest of the image is blank, thus when a symbol is located the entire source image is replaced with the appropriate implementor name, phrase, or entry. For example, the minimum Environment Division would be represented as:

```
*HEADER,COBOL,NC101
     .
     .
     .
001200  SOURCE-COMPUTER.
001300      XXXXX082.
001400  OBJECT-COMPUTER.
001500      XXXXX083.
     .
     .
     .
002100      SELECT PRINT-FILE ASSIGN TO
002200      XXXXX055.
     .
     .
     .
*END-OF,NC101
```

The executive routine, based on information provided by the user, replaces the symbols for the system being validated. The vehicle by which the user describes the implementor defined aspects of COBOL to the PPLVP74 processor is through the use of an "X" parameter card. (The X-card is described in Section 5 including the format and the required information for each of the 84 entries associated with CCVS74.)

The format of the X-card is as follows:

```
X-nnII Text for X-card
```

The 'X-' identifies the parameter as an X-card. The nn is an integer (number) which relates the X-card to a symbol used within the audit routines. The II (optional) represents the starting position within the source image (columns 12-72) in which the current program name or the current job name will be placed if the resolved symbol must be unique within a program, within a job or within the operating system. (If the II is blank, nothing happens.) Note that column 8 of the X-card represents column 12 of the replaced image. The text contained on all X-cards should end with a period.

Using the above example, the information to resolve the above symbols for a few systems could be as follows:

UNIVAC 1100 Series

X-cards		Source Code	
X-55	PRINTER.	001200	SOURCE-PROGRAM.
X-82	UNIVAC-1100.	001300	UNIVAC-1100.
X-83	UNIVAC-1100.	001400	OBJECT-COMPUTER.
			UNIVAC-1100.
		.	
		.	
		.	
		002100	SELECT PRINT-FILE ASSIGN TO
		002200	PRINTER.

IBM OS/VS

X-cards		Source Code	
X-55	UT-S-PRINT.	001200	SOURCE-COMPUTER.
X-82	IBM-370.	001300	IBM-370.
X-83	IBM-370.	001400	OBJECT-COMPUTER.
		001500	IBM-370.
		.	
		.	
		.	
		002100	SELECT PRINT-FILE ASSIGN TO
		002200	UT-S-PRINT.

HIS 6000 GCOS

X-cards		Source Code	
X-55	P1 FOR LISTING.	001200	SOURCE-COMPUTER.
X-82	H6000.	001300	H6000.
X-83	H6000.	001400	OBJECT-COMPUTER.
		001500	H6000.
		.	

```
                    .
                    .
002100      SELECT PRINT-FILE ASSIGN TO
002200      P1 FOR LISTING.
```

DATA GENERAL RDOS

X-cards		Source Code	
X-55	PRINTER.	001200	SOURCE-COMPUTER.
X-82	C-300.	001300	C-300.
X-83	C-300.	001400	OBJECT-COMPUTER.
		001500	C-300.

```
                    .
                    .
                    .
002100      SELECT PRINT-FILE ASSIGN TO
002200      PRINTER.
```

The use of the II (name feature) is described below. This could be useful, for example, if the printer file for each program had to be unique. (In most casts, the select-name for the printer and the associated job control language would be identical for each audit routine.) Another use of the II feature is to make mass storage files unique within the operating system. Consider the following example:

```
*HEADER,COBOL,SQ101
   .
   .
   .
002100      SELECT PRINT-FILE ASSIGN TO
002200      XXXXX055.
   .
   .
   .
007600  FD  SQ-FS1
007700      LABEL RECORDS STANDARD
007800      VALUE OF
007900      XXXXX074
008000      IS
008100      XXXXX075
   .
   .
   .
*HEADER,COBOL,SQ102
   .
   .
   .
003100      SELECT PRINT-FILE ASSIGN TO
003200      XXXXX055.
   .
   .
   .
008200  FD  SQ-FS1
```

```
008300       LABEL RECORDS STANDARD
008400       VALUE OF
008500       XXXXX074
008600       IS
008700       XXXXX075
               .
               .
               .
*END-OF,SQ101
```

The information to resolve the above symbols in source program SQ101 and SQ102 could be as follows:

UNIVAC

```
X-5520 PRINTER XXXXX.
X-74   FILE-ID.
X-7518 "FILE1XXXXX".
```

(Source Program SQ101)

```
               .
               .
               .
002100       SELECT PRINT-FILE ASSIGN TO
002200       PRINTER SQ101.
               .
               .
               .
007600   FD  SQ-FS1
007700       LABEL RECORDS STANDARD
007800       VALUE OF
007900       FILE-ID
008000       IS
008100       "FILE1SQ101"
               .
               .
               .
```

(Source Program SQ102)

```
               .
               .
               .
003100       SELECT PRINT-FILE
003200       PRINTER SQ102.
               .
               .
               .
008200   FD  SQ-FD2
008300       LABEL RECORDS STANDARD
008400       VALUE OF
008500       FILE-ID
008600       IS
008700       "FILE1SQ102"
```

In the above example, the program name (SQ101) was placed in column 20 of the source image (002200) which replaced the symbol XXXXX055. The series of Xs in the X-card has no effect, but is there for purposes of clarity only. The contents of the X-75 control image will result in the current program name being placed in column 18 of the replaced source image. A further example will show a use for the name generated at the job level rather than the program

level. Assume that the program SQ101 passes a file to a subsequent program. The above example could result in a problem in the labeling identification of the file. If the second program (SQ102) refers to the file as 'FILE-ID IS "FILE1SQ102"' and the file was created as 'FILE-ID IS "FILE1SQ101"' the labels would not match. The problem can be resolved in the following way:

```
X-7518  "FILE2JXXXX"
```

The "J" in the position where the first character of the name is to be placed causes the executive routine to use the name of the first or main program instead of the name of the current program.

The following input would result in a slightly different output:

X-cards

```
X-5520  PRINTER XXXXX.
X-74    FILE-ID.
X-7518  "FILE1JXXXX".
```

Source Code

```
(Source Program SQ101)
    .
    .
    .
002100      SELECT PRINT-FILE ASSIGN TO
002200      PRINTER SQ101.
    .
    .
    .
007600  FD  SQ-FS1
007700      LABEL RECORDS STANDARD
007800      VALUE OF
007900      FILE-ID
008000      IS
008100      "FILE1SQ101"
    .
    .
    .
(Source Program SQ102)
    .
    .
    .
003100      SELECT PRINT-FILE ASSIGN TO
003200      PRINTER SQ102.
    .
    .
    .
008200  FD  SQ-FS1
008300      LABEL RECORDS STANDARD
008400      VALUE OF
008500      FILE-ID
008600      IS
008700      "FILE1SQ101"
```

Note that for the printer file each program will still reference a unique file while the name in the VALUE OF clause will be the same "FILE1SQ101" for both programs.

Further refinement of the name placement function permits the use of the relative program number to be used in place of the program name. This number would represent the relative position of the source program after it was selected (e.g., if the first and third programs are selected from the population file, the first program would be 001, the third program would be 002). If a '9' appears in the first position where the first character of the name would be placed, a three digit integer would be placed there, the presence of the letter 'C' will result in a two digit integer being placed there. The program number cannot be used as a job number as the 'J' relates to a job or main program name. It always represents the relative number of the current source program being processed.

7.2 OPTIONAL SOURCE CODE SELECTION

Throughout the audit routines there is source code which has been designated as optional, such that through a control parameter to the executive routine the source code will either be present in the resolved source program or eliminated from the source program altogether. The reason for this is to both expedite the validation process and ensure the accuracy of the results. Some source code is optional at the implementation level (e.g., the VALUE OF clause associated with the File Description entry). Other source code has been designated as optional due to anticipated implementation problems. By requesting the elimination of a given type of source code, the source code is automatically excluded and does not require the updating of each source program containing the code.

The selection of the optional code is covered in Section 6 and needs no further explanation here except as a reminder that it should be taken into consideration when implementing CCVS74.

7.3 JOB CONTROL LANGUAGE RESOLUTION/GENERATION

7.3.1 General Discussion on Alphabet Cards

The alphabet-cards are used to describe a skeleton of the Job Control Language necessary to compile and execute the audit routines. The use and functions of the alphabet-cards are covered in Appendix A of this document and particularly in section A.3 of Appendix A. The purpose of this discussion is merely to present the philosophy of the job control language generation features and provide guidelines and hints as to the use of these features.

-- General Comments -- The name placement feature for the alphabet-cards is similar in nature and scope to that described in the X-card discussion in 7.1 above. The rules are the same for the alphabet-cards as were described for the X-cards. The integer contained in columns 5 and 6 of the PPLVP74 alphabet-card represents the position in the generated control card where the current program name will be placed. The presence of the letter 'J' in the first position where the name is to be placed, results in the main program name being used. The presence of a '9' or 'C' number causes a three or two digit integer representing the relative program number to be placed in the generated control card instead of the program name.

-- Initial and Terminal (I & T) Control Cards -- These control cards are generated prior to the first program selected (I-cards) and after the last program selected (T-cards). These cards are generally not used if

each program or series of related programs is treated as an independent job. If all of the programs to be selected at a given time are to be treated as a single job, then the I and T card would be needed to initiate and terminate the job. The name placement feature is not operative for these two card types because they are only generated once and are therefore unique by definition. The indicator column is also not used for the same reason.

-- Beginning and Ending Control Cards (B & E) -- The Beginning and Ending control cards are generated before and after each source program selected. The control cards, together with the source program, are written to the output file produced by the executive routine. The program name placement is available and is used quite extensively for making each job unique, or uniquely naming files associated with the source and object programs. Since this use is obvious, the main topic here will be the indicator column to control the generation of the Beginning (B) and Ending (E) Job Control Language.

If the indicator column (column 7) is blank, then the control card is generated for all source programs which are not subroutines (CALLed programs) or source library text entries. There are certain job control statements which should be generated at specific times, for example, the job card should be generated only once for each set of programs making up a job. If two programs are related to the degree that the first passes data to the second, they should be contained in the same job. The indicator values are presented below.

blank - Job control statements will be generated before/after each main source program selected but not for source library text entries or subroutines (CALLed programs).

J - Indicates that this is a job level control statement and therefore the control statement would be generated before the first program in a series or after the last program in a series.

L - Indicates that this control statement should be generated before/after a library text entry. It will not be generated for any other source programs selected.

T - Indicates that the control statement should be generated only before/after a program designated as a subroutine (i.e. CALLed program).

Z - Indicates that the control statement should be generated on the VPWORK1 file only. This permits the generation of JCL statements on a separate file.

Combinations - Some of the indicator values produce a combination of the functions described above. These values are supplied since some control statements may be applicable to both main programs and and called programs, or other combinations.

K - Indicates the control statements should be generated as though both the indicator 'J' (job level statement) and the indicator 'L' (library text control statement) were specified, i.e., a job control level library control statement.

S - Indicates the control statement should be generated as though both the indicator blank (non-called programs) and the indicator "T" (CALLed programs) were specified. This can be accomplished by having separate B/E cards which are identical except one has a blank indicator area and the other indicator area contains a "T".

A - Indicates the control statement should be generated before/after all programs/library text entries selected. (As though blank, 'J', 'L', and 'T' were specified.)

7.3.2 Defining the Alphabet-Cards for Job Control Generation

The Job Control Language necessary to compile and execute the audit routines will comprise between 5,000 and 20,000 statements. The general skeleton of Job Control Language is similar for all the audit routines and except for special cases (e.g., SORT, IPC) is identical. Therefore, the PPLVP74 processor contains the capability to generate job control language based on a formal skeletal definition provided through the use of alphabet-cards. A brief example of a job stream shows how the skeletal definitions are produced.

```
     EXEC 8                                    FUNCTION

(1)  @RUN ABCDEF,ACCNT...                      Identifies and establishes a job.
(2)  @ASG,T    TAPE1.,8C,...                   Assigns media facility for program.
(3)  @COB,IBES                                 Invokes the COBOL compiler.
     000100  IDENTIFICATION DIVISION.          Source program.
       .
       .
       .
(4)  @XQT                                      Produces an absolute object module
                                               (from the compiler output), and
                                               executes it.
(5)  @FIN                                      Terminates the job.
```

The 'RUN' statement is the first card in the job. The "ABCDEF" represents the job name and must be unique relative to the other programs/jobs being run. If more than one program must be run together as a job, then the "RUN" statement appears only before the first in a series of programs. (The "RUN statement is the first control card to be generated before a given program and is a job-level control card that is generated only once per independent job.)

The 'ASG' statement is for facility management and is required if or when an audit routine requires any facilities (files) beyond what is normally available at execution time. [This type of statement appears as the second control card (if needed) before the source program. Similar 'ASG' statements must be generated for file assignment for other programs within the job. 'ASG' is therefore a program level control card.]

The 'COB' statement invokes the COBOL compiler. It immediately precedes the source program and is generated for each source program contained in the job. (Program-level control card generated before each program.)

The 'XQT' statement immediately follows the source program and produces an absolute object program which is then executed. (This control card is a program-level control card that must be the first control card following the source program.)

The 'FIN' statement terminates the job. (This is a job-level control card which would be produced only once after the job control statement following the last program in a given job.)

The 'B' and 'E' alphabet-cards are used to describe the job control language shown in the previous example. The 'B' cards are generated before the source program and the 'E' cards are generated after the source program. The order in which the control cards are generated is determined by a two digit number associated with the alphabet-cards. Program-level versus job-level control cards are distinguished by the use of the indicator column which was described above. Job-level control cards are noted by a "J" in the indicator column while program-level control cards are denoted by a blank in the indicator column. The format of the alphabet-cards is:

```
B-nnIIi(Text of B-card)
E-nnIIi(Text of E-card)
```

The nn is the number which controls the order in which the control cards are generated. The II is used for program/job name placement, and the i is the indicator column. The following B, E, and A, alphabet-cards are required to produce the control statements for the above example.

B-0106J@RUN JXXXX,ACCNT...	First card generated, program name placed in control card beginning in column 6 of the generated control card, job level control card to be generated before the first program in a series.
B-02 BREAK***	Special indicator that will cause the facility request control cards to be generated, if needed, for the program being processed.
B-03 @COB,IBES	Last control card to be generated before the source program.
B-04	Blank 'B' card ensuring no further control cards will be generated before the source program.
E-01 @XQT	First control card generated after each program.
E-02 J@FIN	Job level control card to be generated after the last program of a set is processed.
E-03	Blank 'E' card to ensure that no further control cards will be generated.

```
A-01    @AST,T TAPE1.,8C,...
A-02    @ASG,T TAPE2.,8C,...
```

Facility request control cards which will be generated in place of the special indicator 'BREAK***' if these cards are required. There may be any number of 'A' cards depending on the application.

The above skeletal information causes Job Control Language statements to be produced for the initial example shown earlier. This is a very simple case to show how the skeletal information is produced from an example of actual Job Control Language. More than 20 'B' and 'E' cards is not uncommon in implementing the full COBOL Compiler Validation System. There are additional examples and further details about the Alphabet-cards in Appendix A, Section A.3, Alphabet-Cards/Job Control Language Generation (Monitor Section).

7.3.3 Further Definition of File Allocation Job Control Language

The A-card is a specialized alphabet-card that is used to define the file allocation Job Control Language statements associated with the files used by audit routines (i.e., file name in the SELECT statement of the audit routine). The X-card information used to resolve the implementor names associated with each file has a direct relationship to the 'A' card bearing the same number. For example:

X-01 represents the implementor name for a magnetic tape file

A-01 represents the job control statement necessary to allocate the facilities for the file. The 'A' card allocates the file and assumes that it is to be released after the program terminates.

Further, for systems that allocate at the step or activity level rather than the job level:

APO1 allocates the file and assumes that it is to be used by a subsequent program within the current job.

ADO1 is used to allocate a file that is released at the completion of the current program.

Section 5 contains a list of all X-cards used in CCVS74 and identifies those X-cards used for files. The necessary compliment of A-cards must be designed and produced after determining the compiler/operating system requirements.

The presence of the special indicator 'BREAK***' in a B or E card causes the job control language for file allocation to be generated for the files used by the source program being processed.

7.4 PROGRAMS REQUIRING SPECIAL CONSIDERATION

The implementation of the source programs referenced in this section may require additional thought/work on the part of the person implementing the COBOL Compiler Validation System. These programs either require some form of operator interaction, specialized job control language, external data, or

externally produced ASCII data files.

7.4.1 Specialized Job Control Language

There will need to be special consideration in implementing certain modules of CCVS74 due to specialized job control language which may be required.

Library Module

The preparation of the library text which is to be copied by the audit routines making up the library module requires special job control language statements. The library text must be selected [through Plus (+) cards or an environmental request - *ENVIRONMENT K000] and a run stream built which will cause the appropriate library text entries to be created. (See Section 3 for the program and library text names.)

When running the library module audit routines, additional job control language may be necessary to identify the file or directory containing the library text which is to be copied into the source programs at compile time. The discussion of the Library module programs in Appendix D should be reviewed prior to the implementation/running of the library programs.

Sort Module

There are generally additional job control language statements associated with a program using the COBOL sort. These additional statements can be generated automatically by the placement of the special indicator 'SORT****' in the appropriate positions in a 'B' or 'E' card. [This is discussed in 7.3 and Appendix A, A.3.2, B-Cards (E-cards) and A.3.5, S-Cards.] The actual job control language to be generated for a program containing a COBOL sort is defined ated only for sort programs. The 'S' cards are generated if the special indicator 'SORT****' is encountered, the program being processed contains a SORT statement, and there are one or more 'S' cards which have been defined.

Indexed I-O Module

The audit routines for the Indexed I-O module are designed to take into account that some implementations of indexed I-O require two physical files to make up an indexed file (i.e., one file for the data and one file for the index and pointers.) If this is the case, the use of the optional card generation feature permits two implementor names to be contained in the SELECT statecontrol statements ('A' cards).

The lack of a definition in COBOL 74 regarding the maximum length of the RECORD key for an indexed file was also taken into consideration. Through the use of the optional card feature of the VP-routine either an 8 or 32 character RECORD key can be generated.

See Section 5, CCVS74 X-CARDS, (X-44, X-45 and X-46), Section 6, CCVS74 Option Switches (*OPT10 and *OPT11).

7.4.2 External Data

Programs NC109, NC158, and NC204 all require external data when they are executing. This data is accessed by the ACCEPT statement. The data required

is contained in card image formats immediately following the source programs on the population file. If the operating system/compiler permits data which is to be ACCEPTed to be included inline the jobstream controlling the compilation/ execution of a program, this can be accomplished automatically through the use of the PPLVP74 processor using the special indicator 'DATA****' and the optional 'D' cards.

The special indicator 'DATA****' is placed in the appropriate E-card. This is the point within the job control language stream that the data is placed (assuming there is data present following the source program on the population file). If additional job control language is required, the D-01 and D-02 cards be used. The format is the same as for the 'A' cards. The D-01 card will be generated before the data and the D-02 will be generated after data. The 'D' cards are generated only if the data is present following the source program being processed. The data can be suppressed such that it will not be placed in the output file containing the job control language and source programs by omitting the special indicator 'DATA***'.

If the data to be ACCEPTed must be handled differently, the data content to be ACCEPTed is available in Appendix D under the discussion of NC109, NC158, and NC204.

7.4.3 Externally Produced ASCII Files

There is a three program set which comprises an ASCII Validation System. The purpose of these three programs is to validate that the compiler/operating system being tested is capable of processing an ASCII tape file and an ASCII card file produced (in accordance with the appropriate American National Standard) on another system. Also, a magnetic tape and a card file are created in ASCII during the validation. These are taken to another system for further processing to determine whether the compiler/operating system being tested can also produce ASCII files.

The ASCII validation is based on several American National Standards and presumes their support by the compiler/operating system being validated. These are:

1. American National Standard Programming Language COBOL X3.23-1974

   - The CODE-SET clause is used in the File Description entry for ASCII file.

   - The PROGRAM COLLATING SEQUENCE clause is used to process the data in ASCII mode as well as native mode.

   - The SIGN...SEPARATE clause is used for signed data, and all data is defined in the DISPLAY (character) mode.

2. American National Standard Code for Information Interchange (ASCII) X3.4-1968. (Note that this describes the code not the labeling and tape recording formats.)

3. American National Standard Hollerith Punched Card Code, X3.26-1970.

4. American National Standard Magnetic Tape Labels for Information Interchange, X3.27-1969.

5. American National Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NZRI), X3.23-1967.

The 1974 COBOL Standard Language provides the ability to accept, process and produce ASCII code. The ASCII Standard describes what the code looks like insofar as the bit arrangement and configuration, but does not address recording techniques, record formats or any labeling scheme. The 800 density CPI, NZRI magnetic tape recording standard was used to establish the recording density and techniques. (1600 CPI, PE based on X3.39-1973 "Recorded Mangetic Tape for Information Interchange" can be used under special arrangements.) The tape labeling scheme is based on X3.27-1969 but is also compatible with the revision to that tape label standard. Only the VOL1, HDR1, and EOF1 labels are used. The records are fixed length and unblocked.

During an official validation, the Validation Manager for the Federal COBOL Compiler Testing Service will supply the necessary ASCII magnetic tape and card files in addition to the normal tape files associated with a validation. For the ASCII, portion of the validation the following steps are performed:

1. Run SQ118 (SQ119, SQ120)

   SQ118 produces a magetic tape file and card file in ASCII code with the appropriate labels for the system being validated.

   SQ119 reads the magnetic tape and card file to verify their correctness in internal native mode.

   SQ120 further checks the data in internal ASCII mode.

   The magnetic tape file and card file are saved as part of the raw data produced during the validation. These two files are further verified by the Federal COBOL Compiler Testing Service as to the accuracy of the data produced. This is done using a different computer system.

2. Programs SQ119 and SQ120 are rerun using the magnetic tape and card decks supplied by the Validation Manager in place of the two files produced by SQ118. This process checks the ability of the compiler/operating system to read and process ASCII data produced on another computing system.

8. SDD COMPILER VALIDATION PROCEDURES

The procedures used in conducting the validation of a COBOL compiler using one of the U. S. Navy Compiler Validation Systems (CVS) are broken up into several distinct operations:

1) The prevalidation phase includes those activities which must be accomplished prior to the actual validation either in the SDD office or at the validation site.

2) The validation phase is performed at the validation requestor's site and consists of the collection of raw data by the running of the audit routines that make up the CVS.

3) The post-validation phase is performed after the actual validation at the SDD office and consists of analyzing the raw data and the production of a Validation Summary Report (VSR).

The guidelines offered in this section are designed to standardize the validation process and improve the production of the validation team. They are meant to be followed whenever possible.

(DEFINITIONS)

A. Compiler Validation System (CVS) - A system of source programs for a given language which collectively contains the features of that language. The compilation and execution of these programs indicate the degree to which the compiler conforms to its language specifications. There is also an executive routine (PPLVP74) associated with CVS which is used to prepare the source programs for compilation.

B. Source Program Library - This library contains the source programs making up the CVS and the executive system. It is delivered on magnetic tape but can be loaded on mass storage for the duration of the validation. (The COBOL Compiler Validation System identifies this as the "Population File", while the FORTRAN Compiler Validation System documentation identifies this as the FCVS library file.) For the purposes of this document it is referred to as Population File.

C. Compiler Validation System Executive Routine - This program acts as an interface between the validation team and the operating system supporting the compiler being validated. This system is required when doing an offical validation. (The COBOL Compiler Validation System documentation identifies this as PPLVP74 [VP-Routine]; the FORTRAN Compiler Validation System documentation identifies this as the Executive.) For the purposes of this document it is referred to as PPLVP74.

D. SDD - Software Development Division, Federal Operations Directorate, Naval Data Automation Command, Department of the Navy, Washington, D.C., 20376. The SDD supports the Federal COBOL Compiler Testing Service.

E. Validation Team - The validation team is made up of SDD personnel complemented by personnel supplied by the requestor.

F. Validation Manager - SDD team member that is responsible for insuring

that the validation is accomplished according to the official SDD validation procedures. The Validation Manager has the authority to determine when the validation is complete.

8.1 PREVALIDATION IMPLEMENTATION PROCEDURES (SDD)

Certain prevalidation implementation procedures can be accomplished without the use of the target computing system. These are performed prior to the initiation of the validation and are done in the office of the SDD. This permits the Validation Manager to accomplish this part of the validation in his own environment where he has access to information and computer time which may not be available in the installation where the validation is to take place. The information provided by the requestor in supporting software manuals shall be the major source of information for the following steps.

8.1.1 PPLVP74 Parameter Input Preparation

Based on the information provided, the input for the PPLVP74 processor will be prepared to (1) resolve all implementor names within the audit routines (X-cards) and (2) produce the job control language necessary to compile and execute each of the programs (alphabet cards). If there is a computer environmental entry on the Population File for the operating system under which the validation will be run, then any necessary modifications are prepared. If there is no applicable environmental entry then the input information must be produced from scratch.

8.1.2 PPLVP74 Boot Program Deck

A source program must be prepared which will extract the PPLVP74 processor from the Population File and modify it for the target compiler so that it can be compiled for later use during the validation. The information contained in the implementation documentation shall be used in producing or modifying a boot deck for the target system.

8.2 PREVALIDATION IMPLEMENTATION PROCEDURES (ON-SITE)

The following steps should be performed using the target system and must be done prior to the initiation of the validation. The completion of the following procedures is dependent on the successful completion of the prevalidation implementation procedures described in 8.1.1 above.

8.2.1 PPLVP74 Implementation

Using the boot program deck produced in 8.1.2 the PPLVP74 processor is implemented on the target system. The boot deck is compiled on the target system and executed (using the appropriate job control language statements) to extract the PPLVP74 processor and store it in a temporary file for subsequent compilation and cataloging of the object program. Additional changes may have to be made to PPLVP74 based on diagnostics produced by the compiler. There should be no fatal errors generated by the compiler during compilation, but depending on the implementation of the compiler, there may be some syntax which is not acceptable. (This should be noted and kept with the other validation results).

The PPLVP74 source code is modified as necessary and the object program

resulting from its compilation is cataloged (stored) such that it can be executed as needed throughout the validation. The appropriate job control language is produced which will assign (request) the Population File, assign (request) a file for the output of the PPLVP74 processor, and execute the PPLVP74 processor. The method by which the output from the PPLVP74 processor (job control language and source programs) will be introduced to the operating systems must be established. When this has been accomplished the PPLVP74 input parameter data can be tested.

8.2.2 PPLVP74 Input Parameter Test

Once the PPLVP74 processor has been installed and can be executed, the input parameters produced under 8.1.1 are tested. Any corrections/modifications suggested by the requestor as a result of his review are made. A good test for the generated job control language is to select NC101 and SQ101 from the Population File for compilation and execution. The results of the compilations and executions are examined to verify the correctness of the input that has been provided to the PPLVP74 processor.

When the Validation Manager and the requestor's representative agree that the input parameters are correct, a number of copies of the run deck are made. This facilitates the segregation of like programs into separate decks rather than using a single run deck for the entire validation. This means that the changes and options necessary for a given module are contained in a run deck isolated from the information required by the other modules. It also means that several runs can be active at one time, and in the event that the validation is being performed in a closed shop environment, decks can be prepared and submitted at the Validation Manager's leisure without being dependent on the presence of a single run deck. A good practice is to have a run deck for each module and to identify it as such through the job name or run-id.

8.3 VALIDATION PROCEDURES

The objective of a validation is to compile and execute all of the programs required for the level of FIPS PUB 21 (COBOL 68) or FIPS PUB 21-1 (COBOL 74) being validated, or in the case of FORTRAN 77 compile and execute all of the programs contained in the FORTRAN Compiler Validation System. The basic premises upon which a validation is performed are: (1) that all of the programs which are to be used are syntactically acceptable (after modification by the PPLVP74 processor) to the compiler being validated; and (2) that the implementation of the semantic requirements are within the semantic definition of the compiler's language specification. It is assumed that the compiler will syntactically accept all of the program statements and produce the necessary object code to carry out the semantic demands of each of the programs.

If the above are valid premises, then a validation would consist of compiling and executing each program and recording that each program produced the expected results. This is usually not the case, and therefore the following guidelines have been prepared. These guidelines help lead the Validation Manager through a validation in a manner which is consistent with the techniques used in previous validations.

The validation process consists of two phases. The first phase is the syntax check and determines to what degree a compiler accepts the syntax described in its language specifications. The second phase has to do with the

semantic requirements of the language specifications and the degree to which the compiler interprets them correctly.

8.3.1 The Syntax Validation Phase

It is assumed that the audit routines making up the Compiler Validation System are syntactically correct vis-a-vis the language specifications against which the compiler is being validated. Rejection of syntax by the compiler is determined for the most part by the diagnostic messages produced during the compilation process. If there are no serious or fatal diagnostic messages produced and any warning messages provide only expected information (e.g., "MOVE results in truncation" - where truncation is being tested, "Signed subscript" - obviously the subscript cannot contain a negative value but a signed data item can be used as a subscript), then the program is ready for the Semantic Validation phase (8.3.2).

The Validation Manager is responsible for ensuring that all diagnostic messages are evaluated. Diagnostic messages outside the category of "Warning" or "Information" should be eliminated prior to accepting the results of the execution of the object program produced from the compilation of the audit routine. The handling of fatal or serious errors must be done consistently and in a logical manner. The source program which produced the diagnostics must be altered to the extent that it becomes "socially" acceptable to the compiler. Care must be taken in recording the contents of the diagnostic messages in the Compilation/Execution Summary Sheet for that audit routine. This is done to capture information which will be lost once the source program is altered.

Careful consideration must be given to each change being made to an audit routine. If a statement being tested in the Procedure Division (executable statement) is rejected by the compiler, then that portion of the procedural code is changed from executable statements to comments. This causes the source program to compile correctly (with no diagnostic errors) and during execution the test report produced will indicate that the test in question was deleted. (In this case the description of the diagnostic message recorded in the Compilation/Execution Summary Sheet and the statement subsequently eliminated will be used in producing the appropriate entry in the Validation Summary Report.)

When the supporting code produces error diagnostics, then any source code modifications must be planned so that misleading results are not introduced in the report produced during the execution of the audit routine. The worst possible case would be to cause the results of a test to indicate that the test passed when indeed the test was either not executed or the supporting code did not correctly evaluate the results of the test. This applies to supporting code in both data and procedural statements (Data and Procedure Divisions).

Rejection of source code not directly associated with a test can cause problems in the interpretation of the results of a test as well as the interpretation of what to correct and how. Judgement must be exercised such that the purpose of the test is not defeated or that the program is not modified to the extent that a set of tests is no longer relevant.

When a source program has been modified such that no suspicious diagnostic messages are produced, then the execution of the object program is attempted. The execution results of programs which were modified at the source level are examined carefully in the areas where a source language change could impact the

results.

8.3.2 Semantic Validation Phase

The semantic validation phase takes place when the object program produced by the compiler is executed. For the most part the audit routines are self-checking. That is, they execute the code being tested and compare the output of the test against a predetermined result. An output test report is produced which indicates the results of each of the tests contained in a given program. In cases where a discrepancy exists, the output report contains the results produced by the compiler being validated and the predetermined results expected by the audit routine. (There are cases where self-checking is not possible and the output must be visibly examined to determine the results of each of the tests contained in the audit routine.)

The interpretation of the semantic results of the tests is generally straightforward. The source code where the test is located can be found by searching the source program for the definition of the procedure name/test number identified in the output report. If the computed and expected results are available on the output report they can be used in attempting to describe the error. All available information should be recorded on the Compilation/Execution Summary Sheet for that audit routine. (Include the test name, results, an indication of whether a change to the source program could have contributed to the failure, and any other pertinent information.)

8.3.3 Validation Controls

The Validation process is controlled by the Validation Manager. When all of the prevalidation implementation procedures have been completed the validation is ready to begin. There should be a run deck for selecting each module to be tested. (This will vary for COBOL validation depending on the Federal level of COBOL for which the compiler is being validated). Forms are provided to help the Validation Manager maintain controls over the validation and have an up-to-date status of the validation at all times.

- Raw Data Verification Form - This series of documents is used to keep a log of all of the programs which have been run, need to be rerun or have yet to be compiled. The preprinted forms include an entry for each attempt to run the program.

  - Less than successful runs - Information regarding the unsuccessful compilation/execution of a program is recorded. This includes compile time aborts, compile time error diagnostics which prevent execution, execution time aborts, test failures, etc.

  - Execution to Completion

    1. Never achieved - This box is checked if the complete compilation/execution is never achieved. This could be due to a compiler not supporting the language elements being tested. The elimination of all the tests in a given program results in a meaningless execution. During execution of the object program, continued abnormal terminations make the program a candidate for abandonment. The Validation Manager shall determine when a program shall be abandoned.

2. Achieved (No errors) - This is the preferred result. The program compiled correctly and no tests were deleted or produced erroneous results. If this box is checked then the testing of that program is complete. However, if syntax modifications were made to the source program, they are documented in the Compilation/Execution Summary Sheet.

3. Number of failures - This block contains the number of failures indicated on the output report for each audit routine.

4. Number of deleted tests - This block contains the number of deleted tests indicated in the output report for each audit routine.

5. Number of modifications made - This indicates how many source modifications were made to each source program, if any.

The Validation Manager uses the Raw Data Verification form as a check sheet. For any program executed which contains information other than a check in the "achieved - no errors" box, a Compilation/Execution Summary Sheet is produced.

- Compilation/Execution Summary Sheet - This form is used to record additional information about the audit routines which cannot be contained in the Raw Data Verification Form. This summary sheet contains all changes made to the source program and any changes required in the PPLVP74 parameters. Information regarding compile time diagnostics and the solution chosen are also listed. The final information recorded is the execution results. For test failures this includes the computed and correct result (when available) and the test number.

- Information Only Test Log - This is merely a list of information tests (where there is no pass or fail situation) contained in the Compiler Validation System. This list contains the program name or identification of the test. The information is extracted from each of the programs and the test log completed. This will be summarized in the information section of the Validation Summary Report. (Not applicable to COBOL 74.)

All the documents described are used to produce the Validation Summary Report (VSR) for the compiler being validated. Therefore, the more judiciously the forms are filled out, the easier the production of the VSR. If the forms are complete, the majority of the work in producing the VSR can be accomplished by the Secretary/Librarian.

8.3.4 Validation Mechanics

The validation sould be accomplished in an orderly fashion taking advantage of the environment as much as possible. All of the programs must be selected and run at least once. Some programs, due to error diagnostics and abnormal terminations during execution, must be run several times.

The initial pass of the routines can be accomplished without any real validation expertise and could be done by the requestor in the absence of the

validation team. If the on-site implementation can be accomplished by the user, then the validation team need not show up until the first pass of all the programs is complete.

1. After the first pass, the program listings should be segregated into "successful" and "others". After the successful runs have been logged onto the appropriate Raw Data Verification Form, they should be set aside for shipment to the Testing Service office.

2. Programs that compiled correctly but produced erroneous results should be examined in a cursory fashion to verify that the causes of the failure was not due to incorrect input being provided to the compiler/operating system (e.g., switch setting, compile or object time debug switch). If the errors can be attributed to the compiler, they should be logged onto the appropriate Raw Data Verification Form and set aside for shipment to the FCCTS office. Programs whose execution errors can be attributed to incorrect parameter information should be rerun with the correct parameters.

3. Programs that contain source diagnostic messages which are considered to be other than information should be examined to determine what caused the diagnostic message. Source changes should be made in an effort to produce a program that the compiler considers syntactically correct. All changes must be noted on the Compilation/Execution Summary Sheet. The modifications to the source programs are accomplished through the update capability of the PPLVP74 processor. All updates made through the PPLVP74 processor are saved for later verification in producing the VSR.

Programs that have to be modified several times can become troublemakers. The Validation Manager uses his discretion in determining whether to continue experimenting with a program or to declare it as an abandoned program. One must not lose sight of the fact that compiler validation and compiler debugging are not synonymous.

8.4 VENDOR INSTRUCTIONS FOR ACCOMPLISHING A VALIDATION

8.4.1 Program Modifications

All modifications to the source programs must be made through the Executive Routine (PPLVP74) included with the Compiler Validation System. The vendor may use whatever techniques he has available for his own testing requirements, but for an official validation these "other techniques" must be reduced to input to the Executive Routine. All inputs to the Executive Routine must be collected upon completion of the validation for use by the Validation Manager in producing the Validation Summary Report.

8.4.2 Output Requirements

Listings which must be shipped to the FCCTS for further processing include:

a. All output listings produced by the Executive Routine during the validation.

b. The final compilation/execution of each of the source programs

used in the validation process. This includes the compiler produced source listing, any listing produced by the binding process which may be required prior to execution, and the report produced by the execution of the audit routine.

c. The initial compilation/execution of each of the source programs which had to be rerun during the validation process must also be retained.

8.4.3 Shipping Instructions

When the testing is complete all output must be packaged for shipment to the Testing Service for further processing. A packing manifest should accompany each package of listings identifying the name of each of the listings and the relative order of each listing in the package. A duplicate copy of the packing manifests should be provided for the Validation Manager.

The following items are to be shipped to the FCCTS when the collection of raw data is complete:

a. All input to the Executive Routine, in 80 column card form,

b. All output listings produced by the Executive Routine,

c. All source program compilation/execution listings described in 8.4.2 above,

d. All magnetic tapes supplied by the Testing Service. (Copies of the tapes can be saved by the vendor, however the original tapes must be returned.)

The above items should be forwarded to:

Software Development Division
Federal COBOL Compiler Testing Service
Department of the Navy
Washington, D. C. 20376

ATTN: (Use FCCTS control number assigned to the validation)

9. AUDIT ROUTINES FOR EACH LEVEL OF FIPS PUB 21-1 (COBOL 74)

There are four levels making up Federal Standard COBOL. They are Low, Low-Intermediate, High-Intermediate, and High.

9.1 LOW LEVEL OF FEDERAL STANDARD COBOL

The following programs must be compiled/executed for purposes of validating the Low level of Federal Standard COBOL.

MODULE	NUMBER OF PROGRAMS
Nucleus (Level 1)	
NC101-NC120	20
NC151-NC165	15
Table Handling (Level 1)	
TH101-TH111	11
TH151-5H152	2
Sequential I-O (Level 1)	
SQ101-SQ121	21
SQ151-SQ153	3
	72

9.2 LOW-INTERMEDIATE LEVEL OF FEDERAL STANDARD COBOL

The following programs must be compiled/executed for purposes of validating the Low-Intermediate level of Federal Standard COBOL. Note that all of the programs making up the Low Level of Federal Standard COBOL must also be compiled and executed.

MODULE	NUMBER OF PROGRAMS	
(Low Level)		72
Relative I-O (Level 1)		
RL101-RL109	9	
RL151-RL153	3	
RL421	1	
Segmentation (Level 1)		
SG101-SG103 (Exclude SG104-SG106)	3	
SG421	1	
Library (Level 1)		
LB101-LB104 (Exclude LB105)	4	
LB106-LB107	2	
LB421	1	
Debug (Level 1)		
DB101-DB103 (Exclude DB104)	3	
DB105	1	
DB421-DB422	2	
Inter-Program Communication (Level 1)		
IC101-IC123	23	
IC151-IC152	2	
IC421-IC422	2	
	--	
		57

		129

9.3 HIGH-INTERMEDIATE LEVEL OF FEDERAL STANDARD COBOL

The following programs must be compiled/executed for purposes of validating the High-Intermediate level of Federal Standard COBOL. Note that all of the programs making up the lower levels of Federal Standard COBOL must also be compiled and executed.

MODULE	NUMBER OF PROGRAMS	
(Low-Intermediate)		129
Nucleus (Level 2)		
NC201-NC219	19	
NC431	1	
Table Handling (Level 2)		
TH201-TH221	21	
TH431	1	
Sequential I-O (Level 2)		
SQ201-SQ220	20	
SQ431	1	
Relative I-O (Level 2)		
RL201-RL205	5	
RL431	1	
Sort-Merge (Level 1)		
ST101-ST118	18	
ST431-ST434	4	
Segmentation (Level 1)		
SG104-SG106	3	
Library (Level 1)		
LB105	1	
Debug (Level 1)		
DB104	1	
Debug (Level 2)		
DB201-DB202 (Exclude DB203 and DB204)	2	
DB431	1	
Inter-Program Communication (Level 2)		
IC201-IC208	8	
IC431-IC432	2	
Communication Module		
None - Withheld from CCVS74 3.0		
CM431 (for flagging purposes only)	1	
		110
		239

9.4 HIGH LEVEL FEDERAL STANDARD COBOL

The following programs must be compiled/executed for purposes of validating the High level of Federal Standard COBOL. Note that all of the programs making up the lower levels of Federal Standard COBOL must also be compiled and executed.

MODULE	NUMBER OF PROGRAMS	
(High-Intermediate)		239
Indexed I-O (Level 1)		
IX101-IX107	7	
Indexed I-O (Level 2)		
IX201-IX211	11	
IX441-IX442	2	
Sort-Merge (Level 2)		
ST201-ST216	16	
ST441-ST443	3	
Segmentation (Level 2)		
SG201-SG204	4	
SG441	1	
Library (Level 2)		
LB201-LB207	7	
LB441	1	
Debug (Level 2)		
DB203-DB204	2	
		54
		293

9.5 REPORT WRITER MODULE

The FIPS PUB 21-1 excludes the Report Writer Module from the Federal COBOL Standard. However, the Report Writer Module is still tested during a validation if support for that module is claimed by the compiler vendor.

MODULE	NUMBER OF PROGRAMS
(Low through High)	293
RW101-RW104	4
TOTAL CCVS74 V3.0	297

10. AUDIT ROUTINES FOR FLAGGING EACH LEVEL OF FIPS PUB 21-1 (COBOL 74)

There are three groups of audit routines provided for determining the degree to which a given COBOL compiler meets the flagging requirements as defined in FIPS PUB 21-1.

All COBOL elements identified within an audit routine which are above the level of Federal Standard COBOL being syntax checked should be flagged. If a COBOL element above a specified level of Federal Standard COBOL is supported by the compiler, then the message should contain, at a minimum, (1) the identification of the source line number in which the nonconforming syntax occurs, and (2) the identification of the level of Federal Standard COBOL that supports the syntax.

10.1 LOW LEVEL FEDERAL STANDARD COBOL

The following flagging programs must be compiled/executed for purposes of validating the compiler's capability for flagging language elements above the Low Level of Federal Standard COBOL.

MODULE	NUMBER OF PROGRAMS
Communication	
CM431	1
Debug	
DB421-DB422	2
DB431	1
Inter-Program Communication	
IC421-IC422	2
IC431-IC432	2
Indexed I-O	
IX441-IX442	2
Library	
LB421	1
LB441	1
Nucleus (Level 2)	
NC431	1
Relative I-O	
RL421	1
RL431	1
Segmentation	
SG421	1
SG441	1
Sequential I-O (Level 2)	
SQ431	1
Sort-Merge	
ST431-ST434	4
ST441-ST443	3
Table Handling (Level 2)	
TH431	1
Total	26

10.2 LOW-INTERMEDIATE LEVEL FEDERAL STANDARD COBOL

The following flagging programs must be compiled/executed for purposes of validating the compiler's capability for flagging language elements above the Low-Intermediate Level of Federal Standard COBOL.

The designated language elements contained in the following programs must be flagged if the compiler is in conformance with the FIPS PUB 21-1 flagging requirements for the Low-Intermediate level of Federal Standard COBOL.

MODULE	NUMBER OF PROGRAMS
Communication	
CM431	1
Debug	
DB431	1
Inter-Program Communication	
IC431-IC432	2
Indexed I-O	
IX441-IX442	2
Library	
LB441	1
Nucleus (Level 2)	
NC431	1
Relative I-O	
RL431	1
Segmentation	
SG441	1
Sequential I-O (Level 2)	
SQ431	1
Sort-Merge	
ST431-ST434	4
ST441-ST443	3
Table Handling (Level 2)	
TH431	1
Total	19

The designated language elements in the following programs in the Low-Intermediate level of Federal Standard COBOL should not be flagged.

MODULE	NUMBER OF PROGRAMS
Debug	
DB421-DB422	2
Inter-Program Communication	
IC421-IC422	2
Library	
LB421	1
Relative I-O	
RL421	1
Segmentation	
SG421	1
Total	7

10.3 HIGH-INTERMEDIATE LEVEL FEDERAL STANDARD COBOL

The following flagging programs must be compiled/executed for purposes of validating the compiler's capability for flagging language elements above the High-Intermediate Level of Federal Standard COBOL.

The designated language elements contained in the following programs must be flagged if the compiler is in conformance with the FIPS PUB 21-1 flagging requirements for the High-Intermediate level of Federal Standard COBOL.

MODULE	NUMBER OF PROGRAMS
Indexed I-O	
IX441-IX442	2
Library	
LB441	1
Segmentation	
SG441	1
Sort-Merge	
ST441-ST443	3
Total	7

The designated language elements in the following programs are contained in the High-Intermediate level of Federal Standard COBOL and should not be flagged.

MODULE	NUMBER OF PROGRAMS
Communication	
CM431	1
Debug	
DB421-DB422	2
DB431	1
Inter-Program Communication	
IC421-IC422	2
IC431-IC432	2
Library	
LB421	1
Nucleus (Level 2)	
NC431	1
Relative I-O	
RL421	1
RL431	1
Segmentation	
SG421	1
Sequential I-O (Level 2)	
SQ431	1
Sort-Merge	
ST431-ST434	4
Table Handling (Level 2)	
TH431	1
Total	19

10.4 HIGH LEVEL FEDERAL STANDARD COBOL

The following flagging programs must be compiled/executed for purposes of validating the compiler's flagging capability for High Level of Federal Standard COBOL. The designated language elements in these programs are all contained in the High level of Federal Standard COBOL, and therefore, they should not be flagged.

MODULE	NUMBER OF PROGRAMS
Communication	
CM431	1
Debug	
DB421-DB422	2
DB431	1
Inter-Program Communication	
IC421-IC422	2
IC431-IC432	2
Indexed I-O	
IX441-IX442	2
Library	
LB421	1
LB441	1
Nucleus (Level 2)	
NC431	1
Relative I-O	
RL421	1
RL431	1
Segmentation	
SG421	1
SG441	1
Sequential IO (Level 2)	
SQ431	1
Sort-Merge	
ST431-ST434	4
ST441-ST443	3
Table Handling (Level 2)	
TH431	1
Total	26

11. POPULATION FILE FORMAT

The population file source program library is provided on magnetic tape and has the following characteristics*

7 - Track
Even - Parity
BCD - Code
800 BPS - Density

Record size is 2400 characters (30 X 80 character images deblocked by the PPLVP74 processor)

*Tapes received from NTIS may have characteristics other than those described above.

The Population File contains 7286 blocks and 218566 records. The creation date for the population file is 77/09/30*21.46.09. This version identification for the Population File is 3.0 (Version 3 Release 0).

A. PPLVP74 PROCESSOR CONTROL STATEMENTS.

The PPLVP74 processor is an integral part of the software contained in the Program Production Library of the Software Development Division, Federal Operations Directorate, Naval Data Automation Command, and is used to maintain libraries of source programs in a computer independent format. These programs can be rendered computer dependent through the use of the PPLVP74 processor.

The input to PPLVP74 is broken up into two sections. The Monitor Section is used to describe the environment of the computer for which the programs are to be tailored, the type and format of the job control language necessary to compile and execute the programs, and to control the amount of printed output from PPLVP74. The Update Section contains source program updates which are to be applied to the programs as they are selected from the PPLVP74 library. The Update Section also permits the altering of the job control language specified in the Monitor Section. Additionally, new programs can be inserted into the PPLVP74 library in the Update Section.

The commands in the Monitor Section are briefly described below:

*ALTSYSCODE	Causes the any asterisk occurring in column 1 of a source record to be replaced with an alternate system code.
*AUDIT	Causes the appropriate header to be generated for output of each of the audit routines produced for a 1968 CCVS validation.
*BCD-POPFILE *BCD-CONTROL	Used to set a flag to indicate to PPLVP74 whether conversion of the character set of the data contained in the library should be accomplished. (The actual coding which will do the conversion must be provided at the time the PPLVP74 processor is generated.)
*CCVSVERSION	Specifies which, if any, version of the CCVS is being used (68 or 74).
*END-MONITOR	Indicates the end of the Monitor Section and is optionally followed by the Update Section.
*ENVIRONMENT	Requests information which is input to the Monitor Section from the Source Program library (Population File).
*IMCV	Causes columns 76-80 of the Source File to contain a sequence number.
*INITIAL	Initializes tables used by PPLVP74.
*JCL	Controls the generation of JCL statements during the selection of an audit routine.
*LIST	Indicates the type and amount of listing required of PPLVP74.

*NODATA	Causes any data following an *HEADER,DATA* card image to be suppressed when creating a Source File or an updated Population File.
*NOSOURCE	Causes all processing of the Source File to be suppressed.
*OPEN	Causes the VPWORK1 file to be opened and those A, B and E-cards with the 'z' option to be written to this file.
*OPTION	Indicates options which are to be considered in generating source programs for compilation.
*PRGID	Modifies columns 5-6 of the alphabet cards to specify either the current program or the first program in a series of programs.
*RESEQUENCE	Causes all selected audit routine to be resequenced as they are written to the Source File and new Population File.
*SAVE-DELETIONS	Indicates final disposition of optional code in the source programs being selected.
*SOURCE	Controls the information which is written to the Source File.
*SUBSET	Causes a large group of programs to be selected.
*TEST	Causes both the Source File and the new Population File to be suppressed.
*UP-POP	Controls the generation of the new Population File.
ALPHABET Cards	Used to describe job control language requirements and provide implementor defined name for use in COBOL source program.

The Update Section contains updates for the programs being selected and the method of updating is described under the detailed discussion of the Update Section in A.4.

A.1 PPLVP74 COMMANDS (MONITOR SECTION)

The input and control mechanisms accepted by PPLVP74 are a superset of those accepted by PPLVP68. The input to the 1968 VP-routine will function correctly using the 1974 VP-Routine. This permits both the 1968 and 1974 audit routines to be managed using the PPLVP74.

The format of all commands beginning with the control character '*' (asterisk) is flexible to the extent that the command begins in Column 1 and the operands associated with that command follow separated by a comma or one or more spaces. The fixed format input required by PPLVP68 is acceptable but unnecessary. The following is a list of all commands acceptable in the Monitor Section of PPLVP74 and their associated options.

A.1.1 *ALTSYSCODE

The *ALTSYSCODE card causes PPLVP74 VP-Routine to examine every record being written to the Source File, and to replace any asterisk occurring in column 1 of the record with the character specified in the *ALTSYSCODE card. See also the input to PPLVP74.

```
*ALTSYSCODE, X
*ALTSY, X
```

May be used as a Monitor Section control card only.

A.1.2 *AUDIT

The *AUDIT card is necessary for the 1968 CCVS to produce the appropriate header for output of each of the generated audit routines.

```
*AUDIT,X
```

The X option is the only option that will be specified and then only when validating a COBOL compiler with the 1968 CCVS.

A.1.3 *BCD-POPFILE, *BCD-CONTROL

This card indicates that PPLVP74 is to set flags that can be further used, by coding inserted during implementation of PPLVP74, to determine whether code conversion is necessary.

```
*BCD-POPFILE   YES
*BCD-P   NO
*BCD-C,YES
*BCD-CONTROL  NO
```

The flag, set by the processing of the "*BCD" card, can be tested prior to the creating of data and if necessary when code conversion is accomplished.

A.1.4 *CCVSVERSION

This control statement is used to establish the necessary linkage to process the 1968 CCVS, the 1974 CCVS, or in some cases no CCVS but source programs unrelated to compiler validation.

```
*CCVSVERSION,74
*CCVSVR 68
*CCVSVR NONE
```

A.1.5 *END-MONITOR

The *END-MONITOR card is the last card in the Monitor Section and acts as a termination for this portion of the execution of PPLVP74. All control information has been digested by the processor and the commands specified will be carried out. If there are no updates to be applied (*BEGIN-UPDATE) then the next control card should be an end-of-file indicator. See the *BEGIN-UPDATE statement for further information.

```
*END-MONITOR
*END-M
```

A.1.6 *ENVIRONMENT

The *ENVIRONMENT card is used to simulate the inclusion of a group of Monitor Section control cards. The environmental entries are retrieved from the Population File and logically 'added' to the PPLVP74 control card input files. A list of all environment entries (both for JCL/Validation/Job generation as well as program and option requests) is listed in section 3 of this document.

Operands can be separated by one or more spaces or a comma.

```
*ENVIRONMENT    operand-1,operand-2, ..., operand-10

*ENVIRONMENT K0000,LB100,LB200,U1108NAVIC
   (would be equivalent to)
*ENVIRONMENT K0000
*ENVIRONMENT LB100
*ENVIRONMENT LB200
*ENVIRONMENT U1108NAVIC
```

Both of the above examples would result in the same amount of input to the PPLVP74 processor from the environmental entries contained on the Population File. However, the first example would cause the Population File to be read only once whereas the second example would cause the Population File to be reopened after each request has been honored. When multiple environmental entries are requested they are selected as they are encountered on the Population File.

The results of processing the first example by PPLVP74 would be:

```
*ENVIR K0000,LB100,LB200,U1108NAVIC
 COPIED K0000          10 RECORDS
 COPIED LB100           4 RECORDS
 COPIED LB200           4 RECORDS
 COPIED U1108NAVIC    128 RECORDS
```

Unless specifically requested through a *LIST ENVIR statement, the entries will not be listed as they are processed. See the *LIST statement.

A.1.7 *IMCV

The *IMCV card causes columns 76-80 of the Source File to contain a sequence number.

```
*IMCV
*IMCV   YES
*IMCV   NO
```

The generated sequence number begins with 00010 and is incremented by 10. If no operand is specified, YES is assumed. If no IMCV card is processed, then the sequencing does not take place.

The *IMCV card may be located anywhere in the Monitor Section.

A.1.8 *INITIAL

The *INITIAL statement is used to initialize the various tables used by the PPLVP74 processor for storing the contents of the alphabet cards.

```
*INITIAL ALL      (format 1)

*INITIAL Z,Z,Z ...   (format 2)
```

Z may be the letter A, B, D, E, I, S, X, or T.

This is an optional card. When present, it must be in the Monitor Section. More than one *INITIAL card is allowed.

Format 1 causes all 8 alphabet card tables to be filled with blanks immediately.

Format 2 causes the referenced alphabet card table(s) to be filled with blanks immediately.

A format 1 card is redundant if it is the first card in the Monitor Section.

A.1.9 *JCL

```
*JCL SOURCE
*JCL VPWORK1
```

The *JCL card controls the generation of JCL statements during the selection of the audit routine. All JCL statements (alphabet cards) are generated before and after the appropriate programs written to the SOURCE Program file. Some JCL can be alternatively sent to the work file VPWORK1 by using the 'Z' option of column 7 of the alphabet card (i.e. all non 'Z' cards go to the source file while all 'Z' cards go to the VPWORK1 file).

The *JCL card establishes the default file to which the alphabet cards are to be written. The 'Z' cards are then written to the alternate file. The default is as described above and is the same as though the 'SOURCE' option was specified.

When the 'VPWORK1' option is specified then the 'Z' cards are written to the Source File along with the source programs and all other alphabet cards are written to the words file VPWORK1. VPWORK1 can be used as an input to the operating system.

See Section A.3 for a discussion of the alphabet cards and the 'Z' option.

A.1.10 *LIST

The *LIST card controls the amount of update and control information printed by by the VP-Routine.

```
*LIST    operand-1,operand-2, ...
```

The allowable operands are:

Operand Name	Abbreviation
UPDATES	(U)
NOUPDATES	(NOU)
XCARDS	(X)
CTL	(C)
JCL	(J)
INSERT	(I)
ENVIR	(E)
NOENVIR	(NOE)
PROGRAMS	(P)
COMPACT	(COM)
SOURCE	(S)

The PROGRAMS and UPDATES options are mutually exclusive. The JCL and CTL options are equivalent. The default option is UPDATES if no *LIST card is processed. There can be more than one *LIST card present in the Monitor Section and each *LIST card can contain multiple options.

UPDATES (NOUPDATES)-indicates that all updates are to be listed, i.e., OLD and NEW replacements, deleted card images, and inserted card images.

XCARDS - causes all 'XXXXX' card replacements to be listed as they are processed, thereby showing the actual text which replaced each 'XXXXX' card.

CTL - causes the generated JCL to be listed for each program selected (not applicable to PERMANENT updates).

JCL - causes the generated JCL to be listed for each program selected (not applicable to PERMANENT updates).

INSERT - causes *INSERT card images to be listed as they are processed.

ENVIR (NOENVIR)-causes each record retrieved from an environmental entry to be listed as it is processed.

PROGRAMS - causes programs to be listed as they are selected from the Population File. This has no effect on programs being inserted through an *INSERT command.

SOURCE - causes programs to be listed as they are selected from the Population File. This has no effect on programs being inserted through an *INSERT command.

COMPACT - indicates that the printer output should be as compressed as possible, i.e., do not skip to a new page for each program processed.

EXAMPLE *LIST CTL,XCARDS,PROGRAMS

A.1.11 *NODATA

The *NODATA card causes any data following a *HEADER,DATA* card image associated with a selected program to be suppressed when he Source File is built or an updated Population File is created. The absence of the DATA**** entry in one of the ending (E) control cards has the same effect as the *NODATA statement.

*NODATA

This card is used on any system where the assumed method of placing data inline must be modified such that the data is on a different device. The method of placing the data on the alternate device is left to the user. Note, however that the D-cards are released to the Source File in spite of the presence of the *NODATA card.

A.1.12 *NOSOURCE

The *NOSOURCE card causes the opening, writing, and closing of the Source File to be suppressed.

```
*NOSOURCE
```

The NO option of the *SOURCE card has the same effect. The *NOSOURCE card is never used during a validation.

A.1.13 *OPEN

The *OPEN card causes the VPWORK1 file to be opened, and causes A, B and E-cards with the "Z" option to be written to this file.

```
*OPEN       VPWORK1
```

If this card is not present, the "Z" cards will never be released.

A.1.14 *OPT

The OPTION card causes card images on the Population File identified by a code to be deleted when the Source File is created.

```
*OPTn     X     (format 1)

*OPTION
*OPTION10       (format 2)
*OPTION19
```

Format 1 allows one of the 26 option switches to be set.

n is the number 1, 2, through 26.

X is any letter of the alphabet.

Format 2 allows for a switch to be set via one option statement:

```
*OPTION - 1 through 9
*OPTION10 - 10 through 18
*OPTION19 - 19 through 26
```

In the case of format 2 all switches in question must contain a value.

```
*OPTION       A,X,,F
```

This results in the following:

Switch number	Value
1	A

2	X
3	blank
4	F
5 - 9	blank

The default PPLVP74 (Version 3.0) switch settings are as follows:

Switch number	Value
1	A
2	E
3	H
4	L
5	O
6	(blank)
7	Y
8	C
9	(blank)
10	(blank)
11	T
12-26	(blank)

More than one *OPTn card may be present in the Monitor Section.

The following meanings are attached to each number/letter combination for the 1968 CCVS.

Switch number	Value	Comment
1	A	Generate all source statements referring to system switches.
	B	Delete all references to system switches.
2	E	Generate all CLOSE UNIT statements.
	F	Delete all CLOSE UNIT statements.
3	H	Generate all CLOSE REEL statements.
	I	Delete all CLOSE REEL statements.
4	L	Generate references to 'REVERSED' in all OPEN statements.
	M	Delete references to 'REVERSED' in all OPEN statements.

The 1974 CCVS uses the same switches as the 1968 CCVS with the following additions.

Switch Number	Value	Comment

6	X	Generate dump routine to dump all files in I-O audit routines in which one or more of the tests failed.
	blank	Eliminate the dump coding.
7	Y	Write statement for the audit routine report will use the AFTER PAGE and AFTER integer options.
	Z	Write statement for the audit routine report will only use the AFTER integer option.
8	C	Generate the 'VALUE OF' phrase for all File Description entries.
	blank	Suppress the 'VALUE OF' phrase for all File Description entries.
9	G	Generate the optional X card for the VALUE OF phrase in the File Description entry. (X-69)
	blank	X-69 is suppressed.
10	J	Generate the extra X card for indexed I-O files.
	blank	Additional X card is suppressed.
11	T	(Default switch PPLVP74) maximum size key for indexed files (29) characters.
	U	Minimum size key for indexed files (8) characters.

A.1.15 *PRGID

```
*PRGID      xxxxxxx
```

The *PRGID card is used to modify the effect of the number in columns 5-6 of alphabet cards (see A-card, B-card, etc.). The xxxxxxx may be either the word "CURRENT" or "FIRST". Normally the number in columns 5-6 causes the program-id of a main program to appear in the program-id column for all of the associated subprograms, as well as the main program. But if a *PRGID CURRENT card has been encountered, the program-id of the subprogram itself will appear in the program-id column. A *PRGID FIRST card is used to cancel the effects of a previously processed *PRGID CURRENT card and has no other effects.

A *PRGID CURRENT card may apear anywhere in the Monitor Section or in an *INSERT-C table modification. A *PRGID FIRST card may appear only in a table modification.

A.1.16 *RESEQUENCE

The *RESEQUENCE card causes all selected audit routines to be resequenced

as they are written to the Source File and new Population File. This card may appear anywhere in the Monitor Section. It is not used during validations.

To resequence audit routines individually, see Plus-cards (+).

A.1.17 *SAVE - DELETIONS

The *SAVE - DELETIONS control card indicates the final disposition of optional code in the source programs being selected.

```
*SAVE - DELETIONS   YES
*SAVE - DELETIONS   NO
```

The 'YES' option causes any optional code which is not to be compiled to be converted to comment lines (i.e., '*' in Column 7). This results in the sequence numbers of the source programs to be complete instead of missing optional source statements which were not compiled. The default is the NO option.

A.1.18 *SOURCE

The *SOURCE card affects the information which is written to the Source File (SOURCE-COBOL-PROGRAMS).

```
*SOURCE    YES   (format 1)

*SOURCE    NO    (format 2)

*SOURCE    CTL   (format 3)
```

The YES option has effect only when the PERMANENT option of the *BEGIN-UPDATE control card is specified. The effect is to create a Source File which is identical to the New Population File except that it does not contain *HEADER card images, *FILES* card images, or environment data sets.

The NO option causes all OPEN, CLOSE, and WRITE statements for Source File to be bypassed in the VP Routine.

The CTL option causes all COBOL lines of audit routines and environment data sets to be suppresed for the Source File. Thus, the Source File will contain system control language only. The CTL option is the only option of the *SOURCE card that may be used during a validation.

The interaction of the *SOURCE card with the *BEGIN-UPDATE card is illustrated in the decision table below (a hyphen means a card is absent):

*BEGIN-UPDATE card *SOURCE card	YES	NO	CTL	-	PERM YES	PERM NO	PERM CTL	PERM -
Source File created	X		X	X	X		X	
COBOL lines present	X			X	X			
control cards present	X		X	X				
XXXXXnn cards replaces	X		X	X				

A.1.19 *SUBSET

The *SUBSET card causes a large group of programs to be selected.

*SUBSET-X (format 1)

*SUBSET-ALL (format 2)

X is the letter 'A', 'B', 'C', or 'D' representing the four federal levels of COBOL. See FIPS PUB 21 for the 1968 CCVS levels, and FIPS PUB 21-1 for the 1974 CCVS levels.

More than one format 1 card is allowed. Format 1 works in the same manner as a format 1 *ENVIROMENT card, except the groups selected are larger. The results of a *SUBSET-X card may be modified by following with Minus-cards.

Format 2 causes all source programs on the Population File to be selected. The library text data is also selected. Any other selection cards in the input to PPLVP74, including Minus-cards, are ignored.

A.1.20 *TEST

The *TEST card causes both the Source File and the new Population File to be suppressed. It is used only for maintenance operations on the VP Routine and/or to examine the control cards being generated.

*TEST

A.1.21 *UP-POP

The *UP-POP card controls the generation of the new Population File.

*UP-POP xxx

xxx may be either 'YES' or 'NO'.

The *UP-POP card overrides the option specified on the *BEGIN-UPDATE card. The NO option causes the new Population File to be unconditionally suppressed, and the YES option forces the file to be created. However, if PERMANENT is not specified and YES is specified, the XXXXXnn cards on the new Population File will be replaced, while all of the *HEADER, *FILES*, and environmental data

set card images will not be copied.

The interaction of the *UP-POP card with the *BEGIN-UPDATE card is illustrated in the decision table below (a hyphen means the card is absent):

*BEGIN-UPDATE card				PERM	PERM	PERM
*UP-POP card	YES	NO	-	YES	NO	-
Up-Pop File created	X			X		X
*HEADER, etc present				X		X
XXXXXnn cards replaced	X					

A.2 SOURCE PROGRAM SELECTION (MONITOR SECTION)

Program selection is the request for one or more programs (or sets of programs) which are to be operated on, resulting in an updated Population File or a source output file ready for further processing by the operating system.

A.2.1 PLUS-Cards

A Plus-card causes the named program set to be selected and optionally edited in a predefined manner.

```
+ZZZZZ,N,VVV,YYYYY,R
```

The '+' may be used interchangeably with the character 'P'.

ZZZZZ is the five character program name and is the only required field in the Plus-card. The fields are positional in nature and therefore if the second field is not present, its absence must be shown as a null field.

```
PZZZZZ,,VVV
```

N is a local option that will be applied to this program only (see the *OPT card in the Monitor Section).

+RL101,X would result in all optional card images containing an 'X' in column 7 to be selected, column 7 blanked and the card images written to the source output file.

For a permanent update, the contents of the second field are ignored.

VVV represents up to 3 characters which will be used in identifying the new version of the program set selected. In the case of COBOL this is placed in card columns 78-80 of each COBOL image as it is selected.

YYYYY represents the new name by which the progrm will be known. This includes program-id placement in alphabet cards, as well as the updated Population File (if created). A separate Plus-card is necessary to rename the second program in a program set.

R when present indicates that the programs contained in the program set are to be resequenced. (See the *RESEQUENCE Command in the Monitor Section.)

A request for a program set to be resequenced will cause each of the programs in that set to be resequenced. A Plus-card may be present for each of the programs included in the set if any of the aforementioned switches/changes are to be applied. Assuming the program set is made up of:

```
RL101
RL102
RL103
```

```
+RL101,,,,R
```

results in all three programs being selected and resequenced after any updating applied.

```
+RL101,X,1.F,RC101,R
+RL102,X,1.F,RC102,R
+RL103,,,RC103
```

results in the name of each program being changed; the version of RL101 and RL102 being changed to 1.F; local option switch 'X' being applied to RL101 and RL102 and the resequencing of RL101 and RL102.

The Plus-card assumes the presence of a program on the Population File and is not used when programs are "selected" by insertion in the Update Section.

A.2.2 MINUS-Cards

A Minus-card causes a program set which was previously selected by an earlier control card to be deleted from the selction list.

```
-ZZZZZ
```

'-' and 'M' can be used interchangebaly.

ZZZZZ represents the 5 character name of the program set.

This command logically removes the name of the program set from the list of program sets to be processed and applies only to sets of programs, not individual programs included in a set.

A.3 ALPHABET CARDS/JOB CONTROL LANGUAGE GENERATION (MONITOR SECTION)

The alphabet cards are used for two functions by the PPLVP74 processor. The 'X' cards are used to replace implementor names present in COBOL source programs. The other alphabet cards (A, B, D, E, I, S, and T) are used to define the job control language to be generated for each program as well as the order in which the control statements will be generated. The following is a description of the alphabet cards.

X-cards are used to provide implementor defined syntax to be used within source programs. A list of the X-cards required for the 1968 and 1974 COBOL Compiler Validation Systems is included in Section 3.7.

I-cards are produced as the first output from PPLVP74.

B-cards are produced before each program generated and are generated in the order specified in columns 3 and 4 of the B-card. See discussion of the format of the alphabet cards.

E-cards are produced after each program and are generated in the order specified by columns 3 and 4 of the E-card.

T-cards are produced as the last output of PPLVP74.

A-cards represent file assignment/switch setting job control language and are generated when the special indicator 'BREAK***' is found on a B or E card for each program selected.

S-cards represent sort file assignment job control language and are generated when the special indicator 'SORT***' is found in a B or E card for each program.

D-cards are used to sandwich data which follows source programs. The placement of the data and its related control cards is designated by the special indicator 'DATA****' in an E card.

EXAMPLE

```
I-01    First initial job control statement     OPTIONAL
I-02    Second initial job control statement    OPTIONAL

  .
  .
  .
B-01    First job control statement to precede each program
B-02    Second job control statement to precede each program
B-03    Third job control statement to precede each program
  .
  .
  .
E-01    First job control statement to succeed each program.
```

```
E-02    Second job control statement to succeed each program
E-03    Third job control statement to succeed each program
 .                                                          .
 .
 .
T-01    First terminal job control statement    OPTIONAL
T-02    Second terminal job control statement   OPTIONAL
 .
 .
 .
```

The job control statement is punched in the alphabet card as though column 8 was column 1 of the generated control card. There are indicators which suggest when the various job control statements are to be generated and whether unique information is to be placed in each job control statement.

A.3.1 A-Card

A-cards represent operating system assignment instructions for files, switches, and other hardware features. There is a direct relation between the XXXXXnnn, X-cards and A-cards.

```
0        1         2        8
12345678901234567890.....0

A-nnii assignment instruction    (format 1)

APnnii assignment instruction    (format 2)

ADnnii assignment instruction    (format 3)

ASnnii assignment instruction    (format 4)
```

Format 1 - file created and destroyed in this run.
Format 2 - file created in this run and passed to next run.
Format 3 - file created in previous run and destroyed in this run.

nn - corresponds to nn on X-nn cards and XXXXXnnn card images. 01 through 70 for format 1, 01 through 37 for formats 2 and 3.

ii - first column of present program-id for generated assignment card, 01 through 75. If 00 or blank, program-id will not be generated. If column ii of the generated card contains a "J" and the present program is a sub-program, the program-id of the main program will be generated in its place. If column ii of the generated card contains a '9' or a 'C' a 3 or 2 digit sequence number (based on the number of jobs related) will be generated at this point.

Column 8 becomes column 1 of created instruction, column 9 becomes column 2, etc.

A-cards may be placed either in the Monitor Section or in the body of an INSERT-C table modification.

If either format 2, 3, or 4 is requested (because of information on *FILES card image), and is not present, default is to format 1.

Examples:

```
     0         1         2         3         4         5
     12345678901234567890123456789012345678901234567890

1.   A-01    //TAPE1 DD DSN=TAPE1,UNIT=2400
     AP01    //TAPE1 DD DSN=TAPE1,UNIT=2400,DISP=(NEW,PASS)
     AD01    //TAPE1 DD DSN=TAPE1,UNIT=2400,DISP=(OLD,DELETE)
     AS01    //TAPE1 DD DSN=TAPE1,UNIT=2400,DISP =(OLD,PASS)
     X-01    UT-S-TAPE1.

2.   A-01    $       TAPE      T1,x1R
     AP01    $       TAPE      T1,x1SR
     X-01    T1.

3.   A-01    MON$$ MR1,A1
     X-01    TAPE-UNIT MR1.
```

In example 1, all four formats of the A-card are present. In example 2, only formats 1 and 2 are represented. A request for format 3 (ADnn) results in a default to format 1 (A-nn). In example 3, only format 1 of the A-card is present. Any references that require APnn or ADnn results in the A-nn card being generated. The complexity of the A-cards depends on the complexity of the operating system. The PPLVP74 processor can be generated such that it will accept the 'A-' or all 'A' type cards. This may be a factor in the amount of memory required to execute PPLVP74.

A.3.2 B-Card and E-Card

B-cards and E-cards represent operating system control language instructions which are to be placed at the beginning and end of each COBOL program.

```
     0         1         2         3
     1234567 8901234567890123456789012345

     B-kkmmy system language instruction    (format 1)
     E-kkmmy system language instruction

     B-kk  y special indicator              (format 2)
     E-kk  y special indicator
```

kk - numbers, beginning with 01 for first B-card and first E-card, incrementing by 1 for each successive card. The default limit is 20 for the system generation information.

mm - first column of present program-id for created system control instruction, 01 through 75. If 00 or blank, program-id is not generated. If column mm of the generated card contains a "J" and the present program is a subprogram, the program-id of the main program is generated in its place. If column mm of the generated card contains a '9' or a 'C', then a 3 or 2 digit

number is generated. This number is the relative number of the jobs (program sets) being procesed.

y - one of the following characters:

Blank - instruction generated for every audit routine that is not a called program.
A - combines features of "blank", J, L, and T.
J - instruction generated at job level only, i.e., before first program in job or after last program in job. Prevents generation of instruction between programs of data-passing sets, or between calling and called programs.
K - combines functions of J and L.
L - instruction generated for library data-creating program only.
S - combines functions of "blank" and T.
T - instruction generated for called programs only.
Z - instruction generated on VPWORK1 file only. (See A.1.9 the *JCL command for additional information).

Column 8 becomes column 1 of generated instruction, column 9 becomes column 2, etc. The entire alphabet card may be used.

B-cards and E-cards in either Monitor Section, or in the body of an INSERT-C table modification.

Special indicator may be one of the following:

BREAK*** - causes all A-cards which correspond to the *FILES card image(s) of the current program to be released.

SORT**** - causes all S-cards to be released if the current program uses a sort (X in column 71 of the *FILES card image).

DATA***** - causes all D-cards to be released if the current program is followed by an *HEADER,DATA* card image and data. May appear only on an E-card.

Examples:

B and E-Cards

```
0        1         2         3
12345678901234567890123456789012345

(IBM OS/VS)
B-0106J//A91***** JOB,ETC
B-02    //STEPA EXEC COB
B-03    //COB.SYSIN DD *
E-01    /*
E-02    BREAK***
E-03    /*
A-36    //GO.PRINT DD SYSOUT=A
+NC101
```

Generated System Instructions

```
0        1         2         3
12345678901234567890123456789

//A91NC101 JOB,ETC    (example 1)
//STEPA EXEC COB
//COB.SYSIN DD *
000100 IDENTIFICATION DIVISION.
   .
   .  (source program NC101)
   .
/*
//GO.PRINT DD SYSOUT=A
```

```
                                                    /*
(HIS GCOS)
E-0116J$        SNUMB    *****          $        SNUMB    NC101  (example 2)
B-02  J$        IDENT    ACCOUNT,ETC    $        IDENT    ACCOUNT,ETC
B-03   $        COBOL    NDECK          $        COBOL    NDECK
B-04   R        INCODE   IBMC           $        INCODE   IBMC
E-01   $        EXECUTE                 000100 IDENTIFICATION DIVISION.
E-02   $        LIMITS                     .
E-03   BREAK***                           .  (source program NC101)
E-04  J$        ENDJOB                     .
A-36   $        SYSOUT   P1             $        EXECUTE
+NC101                                  $        LIMITS
                                        $        SYSOUT   P1
                                        $        ENDJOB


(UNIVAC EXEC-8)
B-0108J@RUN    ABCDE,ACCT,ETC           @RUN    NC101,ACCT,ETC
B-02    BREAK***                        @ASG,T TAPE1,8C.,12345
B-03    @COB,BESI                       @COB,BESI
E-01    @XQT                            000100 IDENTIFICATION DIVISION.
E-02    @FIN                               .
A-36    @ASG,T TAPE1,8C.,12345             .  (source program NC101)
+NC101                                     .
                                        @XQT
                                        @FIN
```

A.3.3 D-Card

D-cards represent operating system control language that precedes, follows, or surrounds in-line data.

```
0         1         2
12345678901234567890

D-kkmm system language instruction
```

kk - may be either 01 or 02.

mm - first column of present program-id for created system language instruction, 01 through 75. If 00 or blank, program-id is not generated.

Column 8 becomes column 1 of generated instruction, column 9 becomes column 2, etc.

The D-01 card, if present, is released in front of the data, and the D-02 card, if present, is released behind the data.

A.3.4 I-Card

I-cards represent the system control language needed at the initial position of a jobstream.

```
0         1         2         3
123456789012345678901234567890123456

I-kk  system language instruction
```

kk is a number beginning with 01 for the first I-card, incrementing by 1 for each successive card. The default limit is 10 and may be modified at generation time.

Column 8 becomes column 1 of generated instruction, column 9 becomes column 2, etc. The entire card may be used.

The I-cards are released immediately before the B-cards of the first selected program. I-cards are entered into the VP Routine as Monitor Section control cards.

A.3.5 S-Card

S-cards represent the operating system control language instructions necessary when a COBOL program contains the SORT verb.

```
0         1         2
12345678901234567890

S-kkmm system language instruction
```

kk - numbers, beginning with 01 for first S-card, incrementing by 1 for each successive card. The default limit is 5 and can be modified at generation time.

mm - first column of present program-id for created system control instruction, 01 through 75. If 00 or blank, program-id is not generated. If column mm of the generated card contains a "J" and the present program is a subprogram, the program-id of the main program is generated in its place.

Column 8 becomes column 1 of generated instruction, column 9 becomes column 2, etc.

S-cards may be placed either in Monitor Section, or in the body of a table modification.

Examples:

```
     0         1         2         3         4
     12345678901234567890123456789012345678901

1.   S-01    //SYSWK1 DD DSN=SYSWK1,UNIT=2400
     S-02    //SYSWK2 DD DSN=SYSWK2,UNIT=2400
     S-03    //SYSWK3 DD DSN=SYSWK3,UNIT=2400

2.   S-01    $       FILE    S1,X1R
     S-02    $       FILE    S2,X2R

3.   S-01    @ASG,T XA.,F///300
     S-02    @ASG,T XB.,F///300
```

A.3.6 T-Card

T-cards represent the system control language needed at the terminal position of a jobstream.

```
          0         1         2         3
          123456789012345678901234567890

          T-kk   system language instruction
```

kk is a number beginning with 01 for the first T-card, incrementing by 1 for each successive card. The default limit is 5 and can be modified at generation time.

Column 8 becomes column 1 of the generated instruction, column 9 becomes column 2, etc. The entire card may be used.

The T-cards are released immediately after the E-cards of the last selected program. T-cards are entered into the VP Routine as Monitor Section control cards.

A.4 PPLVP74 COMMANDS (UPDATE SECTION)

The following PPLVP74 commands are used in the Update Section and are used in inserting new data sets/programs into a Population File and also in making changes to existing source programs.

The Update Section begins with the *BEGIN-UPDATE card and terminates with the *END-UPDATE card. There are two additional control cards recognized in the Update Section.

*START (precedes update cards)

*INSERT (precedes a data set to be inserted)

There are several options which enable a variety of tasks to be accomplished.

A.4.1 *BEGIN-UPDATE

The *BEGIN-UPDATE card is the first card of the Update Section and must immediately follow the *END-MONITOR card.

```
*BEGIN-UPDATE                    (format 1)

*BEGIN-UPDATE    PERMANENT       (format 2)
*BEGIN, PERMANENT

*BEGIN-UPDATE    INITIALIZE      (format 3)

*BEGIN-UPDATE    RELATIVE        (format 4)
```

Only 1 operand can be specified in the *BEGIN-UPDATE card.

Format 1 causes the Population File to be opened and assumes that programs are being selected for compilation/execution.

Format 2 causes both the Population File and the new Population File to be opened. All alphabet cards are ignored and XXXXXnnn cards are not replaced. All updates and insertions which follow are applied to the new Population File. This option should not be used for a validation.

Format 3 causes PPLVP74 to assume that a new Population File is being created from scratch. The absence of the Population File (i.e., not assigned) will cause the same assumption.

Format 4 is not used.

A.4.2 *INSERT

The *INSERT function allows information to be introduced to the PPLVP74 processor in the Update Section.

```
*INSERT     aaaaaa,zzzzz,SYSCODE,x   (format 1)
```

```
*INSERT-C    aaaaaa,zzzzz                (format 2)

*INSERT-P    aaaaaa,zzzzz                (format 3)

*INSERT-D    aaaaaa,zzzzz                (format 4)

*INSERT-B    aaaaaa,zzzzz                (format 5)
```

aaaaaa is either the word "FIRST", "BEFORE", "AFTER", or "LAST".

zzzzz is a five-character program-id. It must be present when "BEFORE/AFTER" is used, and is ignored when "FIRST/LAST" is used.

SYSCODE is optional. When present, it causes the PPLVP74 processor to read the next non-blank character and use this as an alternate system code for this insertion. (This permits system JCL to be read and coded with an '*' asterisk in column 7 which is replaced by the specified system code as it is written out to the Source File.)

All formats of the *INSERT card are always followed by one or more cards called the body of the insertion (or the body of the table modification in format 2), which is in turn followed by an *END-INSERT card. The *INSERT card, the body, and the *END-INSERT card, taken collectively, is called the insertion or the table modification. The body of a format 1 insertion consists of data cards or operating system control cards with the character in column 1 replaced by an asterisk. (This is to disguise the system control card such that it can be read through the system input file.) The character normally used in column 1 is punched in the *INSERT card. When the BEFORE/AFTER option is used, the inserted cards are written to the Source File immediately before the B-Cards of the program designated by zzzzz (zzzzz need not be selected) or after the E-cards of the program designated by zzzzz. When the FIRST/LAST option is used, the inserted cards are written after the I-cards or after the E-cards of the last selected program.

The body of a format 2 table modification consists of alphabet cards and a limited number of PPLVP74 control cards. These cards are not written to any file; each card is immediately processed as if it had been read in the Monitor Section. The following are the allowable control cards:

```
*OPTION
*PRGID
*LIST
*BCD-POPFILE
*BCD-CONTROL
Alphabet cards (except 'I' which would have no meaning)
```

Format 3 insertions are generally not used for validations, and may be used in conjuction with any of the options of the *BEGIN-UPDATE card. The body of the insertion consists of a source program or library data set, including the *HEADER and *FILES cards. The program or data set is sequenced (incrementing by 100) and stored in the updated Population File or written to the source output file.

Format 4 is used to insert data which logically follows a source program. If the data exists on the Population File then this permits the data to be

logically replaced by the text within the *INSERT-D, *END-INSERT brackets.

Format 5 is used to create source programs from a library of kernels and the resultant output written either to the Population File or the Source File.

Insertions and table modificatons are placed in the Update Section, alternating with update groups as appropriate, in the order they are to be used. It is permissable for several insertions and table modifications to appear consecutively.

A.4.3 *END-INSERT

An *INSERT or *INSERT-n card is terminated only when an *END-INSERT card is encountered. Any *START, *INSERT or *END-UPDATE cards encountered prior to an *END-INSERT card image are treated as data to be inserted.

A.4.4 *START

The *START card equates a set of updates to a program being retrieved from a Population File. The format for the *START card is as follows:

```
*START,ZZZZZ
```

ZZZZZ represents the name of the program which is to be updated.

If the named program is selected (see the 'P' or '+' card), then the update following the *START card is applied. Any sequence errors detected are flagged. The updates must appear in ascending order based on the line numbers associated with the program being updated. The *START cards and their associated updates must appear in the order in which the programs are found on the Population File.

The *HEADER and *FILES cards can also be updated. The changed card images must appear immediately following the *START card in the order shown:

```
*START,NC101                    *START,FM003
*HEADER,COBOL,NC101             *HEADER,FORTR,FM003
*FILES1,COBOL,NC101             *FILES1,FORTR,FM003
*FILES2,COBOL,NC101                 .
*FILES3,COBOL,NC101                 .
Source program updates          Source program updates
```

The updates for a given program are terminated by another *START card, an *INSERT card or an *END-UPDATE card. The method for updating both COBOL and FORTRAN programs is in the following sections:

A.4.4.1 COBOL Source Program Updating

1. Replace a card image

```
*START (update)                 POPULATION FILE
001900 text of update           001900
```

If the sequence numbers match, the card image is replaced. (Should there not be a card numbered 001900 on the Population File, the card would be inserted.)

2. Delete a card image.

   021900, (rest of card blank) 021900

   The corresponding card is deleted. If there is no corresponding card, an error message is issued.

3. Delete a series of card images.

   first,last
   013900,014100

   The card images from 013900 through and including 014100 are deleted.

4. Insert a card image

   021910 text of card 021900
   022000

   The card image is inserted between 021900 and 022000. (If the card number matched a card number already contained in the program then it would replace that card image.)

   Any blank line numbers are immediately inserted following the action taken on the last numbered card.

   021900,023000
   (Blank) text
   (Blank) text

The two blank cards are inserted after the last card was deleted based upon the request to delete the card images 021900 through 023000.

Characters other than blank or comma (,) can be used in position 7 of the update card:

1. Replacement or insertion of special character cards.

   '-' has no effect on the update. The '-' will appear in the COBOL source card and will be assumed to be associtated with character-string continuation.

   '*' has no effect on the update. The '*' will appear in the COBOL source and will be treated by a COBOL compiler as a comment card.

   'D' - If the PPLVP74 processor is working in the 74 mode (*CCVSR 74), the 'D' has no effect and appears as a debug line in the updated program. If PPLVP74 is working in the 68 mode (*CCVSR 68), then the card image is treated as an optional card (see A-Z).

'A' - 'Z'. The presence of these characters causes the card image to become an optional card. The characters will appear in Column 7 of the population file card image, but the card must be selected via an *OPTION command if it is to be passed to the source output file (see the *OPT card under the discussion of the monitor section). If the character is selected as an option, then each card image containing the character in column 7 has column 7 blanked before the card image is written to the source output file. If the character in Column 7 is not selected then the card is not written to the source output file.

For an update in which a permanently updated Population File is being created, column 7 is not a criterion determining whether the card image is written to the updated (new) Population File. (The card image will always be written to the Population File.)

2. Series operation (001900?002000)

The character in column 7 is placed in column 7 of each of the card images contained in the identified series.

```
001900A002000                001900A
                             001910A
                             002000A
```

'-' The use of the dash is meaningless and results in a program that would, more likely than not, be syntactically incorrect.

'*' This causes a series of card images to become comment entries. This should be used with care or a 'portion' of a of a statement will become a comment resulting in a syntactically incorrect program. This is used during validations to eliminate code rejected by the compilers. As such the code becomes comments and is preserved for use in preparing the Validation Summary Report.

'A' - 'Z', This causes a series of card images to become optional card images (or debug lines 'D'). This should be used with care for the same reasons as stated above under the '*'. See the previous discussion about optional cards under replacement and insertion of special character cards.

A.4.4.2 Source Program Updating for FORTRAN, PL/I, ALGOL, and Other Languages Where the Sequence Numbers are in Columns 73-78.

1. Replace a card image.

```
*START (update)              POPULATION FILE
=00190,00190                 Source text                    001901.0
  (Text of replaced card image follows)
```

If the sequence numbers match, the card image is replaced. (Should there not be a card numbered 00190 on the Population

File for that source program, the card would be inserted.)

2. Delete a card image.

```
=02190,02190                                              021901.0
  (Followed by another =-card or an *-card)
```

The corresponding card is deleted. If there is no corresponding card, an error message is issued.

3. Delete a series of card images.

```
first,last
=01390,01410
  (Followed by another =-card or *-card)
```

The card images from 01390 through and including are deleted.

4. Insert a card image.

```
=021900                                                   021901.0
                                                          022001.0
  (Card images to be inserted follow this card)
```

The card images would be inserted between 02190 and 02200.

The use of characters other than blank or comma (,) in position 7 of the update card:

1. Replacement or insertion of special character cards.

'A' - 'Z' The presence of these characters causes the card image to become an optional card. The characters will appear in column 1 of the Population File card image. The card must be selected via an *OPTION command if it is to be passed to the source output file (see the *OPT card under the discussion of the Monitor Section). If the character is selected as an option, then each card image containing the character in column 1 has column 1 blanked before the card image is written to the source output file. If the character in Column 1 is not selected then the card is written to the source output file and Column 1 contains a 'C'.

For an update in which a permanently updated Population File is being created, column 1 is not a criterion determining whether the card image is written to the updated (new) Population File. (The card image wll always be written to the Population File.)

2. Series operation (=00190?00200)

The character in column 7 is placed in column 1 of each of the card images contained in the identified series.

```
=00190A00200                                              001901.0
                                                          001911.0
```

002001.0

'A' - 'Z' This causes a series of card images to become optional card images. See the previous discussion about optional cards under replacement and insertion of special character cards.

A.4.5 *END-UPDATE

*END-UPDATE

The *END-UPDATE card is an indicator to the PPLVP74 program that the Update Section is to be terminated. It signals the end of all updates, table modifications, and insertions, and causes all files to be closed at the appropriate time.

B. PPLVP74 Generation

The PPLVP74 processor can be implemented either through the use of the bootstrap method or through a system generation technique. The bootstrap method is accomplished through the use of the source program described in section B.1 and can be used to extract the PPLVP74 source program for further compilation. The system generation technique can be used if the PPLVP74 process has been previously implemented. (The current or any previous version of PPLVP74 can be used.)

The PPLVP74 processor is contained within the population file and can be extracted using the provided 'boot' program, the 1968 VP-Routine or PPLVP74 itself once it has been implemented. The length of the internal tables used by PPLVP74 are adjustable at compile time and can be modified to accomodate as many or as few entries as are necessary. For example, if no Initial control cards are needed then only one (1) occurrence for the I table would be requested. If a larger number were needed then the number specified would become the number of occurrences for that table.

B.1 Bootstrap Implementation

The bootstrap program will extract the PPLVP74 processor in such a way that the values specified in the VALUE clause for the following data items will be used for each table whose length can be varied. The user must provide implementor names in the entries named below where an implementor name is requested; and if a default value is to be changed, the appropriate data item VALUE clause must be changed.

CTL-01 Implementor-name used in the SELECT statement for the population file.

CTL-02 Implementor-name used in the SELECT statement for the output file from PPLVP74.

CTL-03 Implementor-name used in the SELECT statement for a sequentially organized work file.

CTL-04 (Same as CTL-03).

CTL-10 Number of entries to be associated with the AP table - 58 default.

CTL-11 Number of entries to be associated with the AD table - 58 default.

CTL-12 Number of entries to be associated with the A table - 58 default.

CTL-13 Number of entries to be associated with the I table (Initial control cards) - 10 default.

CTL-14 Number of entries to be associated with the B table (Beginning control cards) - 20 default.

CTL-15 Number of entries to be associated with the E table

(Ending control cards) - 20 default.

CTL-16 Number of entries to be associated with the T table (Terminal control cards) - 5 default.

CTL-17 Number of entries to be associated with the X table 90 minimum required for the 1974 CCVS - 150 default.

CTL-18 Number of entries to be associated with the S table (Sort control cards) - 5 default.

CTL-19 Number of entries associated with the program selection table - default 260.

CTL-36 Implementor-name used in the SELECT statement for a printer destined file.

CTL-37 Implementor-name used in the SELECT statement for the card reader.

CTL-45 Implementor-name associated with the top of page mnemonic-name to be used with a WRITE statement for a printer destined file.

CTL-48 OMITTED or STANDARD, used in conjunction with the LABEL RECORD clause associated with the printer destined file produced by PPLVP74.

CTL-49 Name of the Source-Computer.

CTL-50 Name of the Object-Computer.

The following is a source COBOL program which will extract the PPLVP74 processor for further compilation and implementation on the system on which the CCVS is being implemented. (It can also be used to extract the 1968 VP-Routine associated with the 1968 CCVS.)

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    VPBOOT.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT    POPFILE ASSIGN TO
       (implementor name for the input tape)
    SELECT  PRT ASSIGN TO
       (implementor name for the printer)
    SELECT  VP-FILE ASSIGN TO
       (implementor name for the file to which the PPLVP74
        processor source code is to be placed.)
I-O-CONTROL.
DATA DIVISION.
```

```
FILE SECTION.
FD POPFILE
    LABEL RECORDS OMITTED
    DATA RECORD POP-REC
    BLOCK CONTAINS 1 RECORDS
    RECORD CONTAINS 2400 CHARACTERS
    01  POP-REC.
    02 CARD1 PIC X(80) OCCURS 30 TIMES.
FD  VP-FILE
    LABEL RECORDS STANDARD
    DATA RECORD IS VP-REC.
01  VP-REC.
    02 VP-1-72.
        05 VP-1-6 PIC X(6).
        05 VP-7-7 PIC X.
        05 VP-8-11 PIC X(4).
        05 VP-12-72.
            07 VP-12-18.
            09 VP-12-16 PIC XXXXX.
            09 VP-17-19 PIC 99.
            88 X01 VALUE 001.
            88 X02 VALUE 002.
            88 X03 VALUE 003.
            88 X04 VALUE 004.
            88 X10 VALUE 010.
            88 X11 VALUE 011.
            88 X12 VALUE 012.
            88 X13 VALUE 013.
            88 X14 VALUE 014.
            88 X15 VALUE 015.
            88 X16 VALUE 016.
            88 X17 VALUE 017.
            88 X18 VALUE 018.
            88 X19 VALUE 019.
            88 X36 VALUE 036.
            88 X37 VALUE 037.
            88 X45 VALUE 045.
            88 X48 VALUE 048.
            88 X49 VALUE 049.
            88 X50 VALUE 050.
            07 FILLER PIC X(54).
    02 VP-73-77 PIC X(5).
    02 VP-78-80 PIC X(3).
FD  PRT
    LABEL RECORDS STANDARD
    DATA RECORD PRT-REC.
01  PRT-REC.
        02 FILLER PIC X(130).
WORKING-STORAGE SECTION.
01  SUB1 PIC S9(9) USAGE COMP SYNC RIGHT VALUE 55.
01  VP-REC-HOLD PIC X(80).
01  CONTROL-CARDS.
    03 CTL-01 PIC X(28) VALUE "population file select.".
    03 CTL-02 PIC X(28) VALUE "PPLVP74 output select. ".
    03 CTL-03 PIC X(28) VALUE "workfile select.       ".
```

```
           03 CTL-04 PIC X(28) VALUE "workfile select.       ".
           03 CTL-10 PIC X(28) VALUE "58.                    ".
           03 CTL-11 PIC X(28) VALUE "58.                    ".
           03 CTL-12 PIC X(28) VALUE "58.                    ".
           03 CTL-13 PIC X(28) VALUE "10.                    ".
           03 CTL-14 PIC X(28) VALUE "30.                    ".
           03 CTL-15 PIC X(28) VALUE "50.                    ".
           03 CTL-16 PIC X(28) VALUE "05.                    ".
           03 CTL-17 PIC X(28) VALUE "150.                   ".
           03 CTL-18 PIC X(28) VALUE "05.                    ".
           03 CTL-19 PIC X(28) VALUE "260.                   ".
           03 CTL-36 PIC X(28) VALUE "printer select.        ".
           03 CTL-37 PIC X(28) VALUE "card-reader select.    ".
           03 CTL-45 PIC X(28) VALUE "advancing top of page. ".
           03 CTL-48 PIC X(28) VALUE "STANDARD or OMITTED    ".
           03 CTL-49 PIC X(28) VALUE "source computer name.  ".
           03 CTL-50 PIC X(28) VALUE "object computer name.  ".
       PROCEDURE DIVISION.
       DRIVER-SECTION SECTION.
       DRIVER1.
           OPEN     INPUT POPFILE.
           OPEN     OUTPUT PRT VP-FILE.
           MOVE "START" TO PRT-REC.
           WRITE PRT-REC.
      *    *** DOWHILE VP-1-72 NOT EQUAL TO . . .
           PERFORM READ-POP UNTIL VP-1-72 EQUAL TO
                   "*HEADER,COBOL,ASVP9    ".
      *    NOTE THAT "*HEADER,COBOL,ASVPO    " WOULD SELECT THE SMALLER
      *           EXECUTIVE ROUTINE.
      *     ENDDO
           WRITE    PRT-REC FROM VP-REC.
      *    *** DOUNTIL VP-1-6 EQUAL TO . . .
           PERFORM SELECT-VP.
           WRITE    PRT-REC FROM VP-REC.
           PERFORM SELECT-VP UNTIL
               VP-1-6 EQUAL TO "*END-O" OR
               VP-1-6 EQUAL TO "*HEADE".
      *    ENDDO
           WRITE    PRT-REC FROM VP-REC.
           MOVE  "FIN    " TO PRT-REC.
           WRITE PRT-REC.
           CLOSE PRT VP-FILE POPFILE.
           STOP RUN.
       SELECT-VP SECTION.
       SLE-01.
           PERFORM READ-POP.
           IF  VP-1-6 EQUAL TO "*HEADE" OR
               VP-1-6 EQUAL TO "*FILES" OR
               VP-1-6 EQUAL TO "*END-O"
                   GO TO SELECT-EXIT
           ELSE
               PERFORM PROCESS-VP
               IF  VP-7-7 EQUAL TO " " OR
                   VP-7-7 EQUAL TO "-" OR
                   VP-7-7 EQUAL TO "*"
```

```
            PERFORM WRITE-CARD
        ELSE
            IF VP-7-7 EQUAL TO "A" OR
               VP-7-7 EQUAL TO "E" OR
               VP-7-7 EQUAL TO "C" OR
               VP-7-7 EQUAL TO "L" OR
               VP-7-7 EQUAL TO "I" OR
               VP-7-7 EQUAL TO "J"
               MOVE SPACE TO VP-7-7
               PERFORM WRITE-CARD.
*The "J" above causes the source output file of PPLVP74
*to contain 80 character records.  The absence of that statement
*would result in 72 character records for that file.
*           ENDIF
*       ENDIF
*   ENDIF
 SELECT-EXIT.
 READ-POP SECTION.
 READ1.
     IF      SUB1 GREATER 30
             MOVE  1  TO SUB1
             PERFORM READP1.
     MOVE CARD1 (SUB1) TO VP-REC
     IF     VP-1-6 EQUAL TO "*HEADE"
     WRITE    PRT-REC FROM VP-REC.
     ADD  1 TO SUB1.
 READP1 SECTION.
 RDD.
     READ POPFILE AT END MOVE "*END-O" TO VP-1-6.
 PROCESS-VP SECTION.
 PROC1.
     IF      VP-12-16 EQUAL TO "XXXXX" AND
             VP-17-19     NUMERIC
             NEXT SENTENCE ELSE GO TO PROCEND.
     IF      X01 MOVE  CTL-01 TO VP-12-72 GO TO PROCEND.
     IF      X02 MOVE  CTL-02 TO VP-12-72 GO TO PROCEND.
     IF      X03 MOVE  CTL-03 TO VP-12-72 GO TO PROCEND.
     IF      X04 MOVE  CTL-04 TO VP-12-72 GO TO PROCEND.
     IF      X10 MOVE  CTL-10 TO VP-12-72 GO TO PROCEND.
     IF      X11 MOVE  CTL-11 TO VP-12-72 GO TO PROCEND.
     IF      X12 MOVE  CTL-12 TO VP-12-72 GO TO PROCEND.
     IF      X13 MOVE  CTL-13 TO VP-12-72 GO TO PROCEND.
     IF      X14 MOVE  CTL-14 TO VP-12-72 GO TO PROCEND.
     IF      X15 MOVE  CTL-15 TO VP-12-72 GO TO PROCEND.
     IF      X16 MOVE  CTL-16 TO VP-12-72 GO TO PROCEND.
     IF      X17 MOVE  CTL-17 TO VP-12-72 GO TO PROCEND.
     IF      X18 MOVE  CTL-18 TO VP-12-72 GO TO PROCEND.
     IF      X19 MOVE  CTL-19 TO VP-12-72 GO TO PROCEND.
     IF      X36 MOVE  CTL-36 TO VP-12-72 GO TO PROCEND.
     IF      X37 MOVE  CTL-37 TO VP-12-72 GO TO PROCEND.
     IF      X45 MOVE  CTL-45 TO VP-12-72 GO TO PROCEND.
     IF      X48 MOVE  CTL-48 TO VP-12-72 GO TO PROCEND.
     IF      X49 MOVE  CTL-49 TO VP-12-72 GO TO PROCEND.
     IF      X50 MOVE  CTL-50 TO VP-12-72 GO TO PROCEND.
 PROCEND. EXIT.
```

```
WRITE-CARD SECTION.
SWRITE.
    MOVE SPACE TO VP-73-77 VP-78-80.
    WRITE    VP-REC.
```

B.2 PPLVP74 System Generation

The PPLVP74 processor can be selected and generated by using the 1968 VP-Routine or the 1974 VP-Routine. The information to be provided parallels the information required by the 'boot' program described above. There are additional options which can be used to configure the PPLVP74 processor to a particular environment. The options designated as "Default" should be used to produce a PPLVP74 processor identical to the one used by the Federal COBOL Compiler Testing Service.

OPTIONS

*OPT1	A	(Default) Unsegmented program.
	B	Segmented COBOL program.
*OPT2	E	(Default) Include all AP tables and appropriate references.
	(BLANK)	AP tables and associated references will not be included.
*OPT3	C	(Default) Include all AD tables and appropriate references.
	(BLANK)	AD tables and associated references will not be included.
*OPT4	I	(Default) Include 1968 CCVS XXXXXNN numbers for implementor-names.
	H	Include 1974 CCVS XXXXXnnn numbers for implementor-names.
*OPT5	L	INSPECT
	K	EXAMINE
*OPT7	J	(Default) Source program output file will contain 80 character records.
	(BLANK)	Source program output file will contain 72 character records.

X-Card Parameters - X numbers are given for both the 1968 and 1974 CCVS - *OPT4 should be set to I for 1968 (CCVS68) X-number or set to H for 1974 (CCVS74) X-numbers.

Note that X-10 through X-19 have to be provided regardless of whether the 1968 or 1974 X-cards are used.

1968	1974	Contents
X-01	X-01	Implementor-name for SELECT statement for the Population File.
X-02	X-02	Implementor-name for SELECT statement for the Output from PPLVP74.
X-03	X-03	Implementor-name for SELECT statement for the Sequential Work-

file.

X-04	X-04	Implementor-name for SELECT statement for the Sequential Work-file.
	X-10	Number of entries to be associated with the AP table. (58)
	X-11	Number of entries to be associated with the AD table. (58)
	X-12	Number of entries to be associated with the A table. (58)
	X-13	Number of entries to be associated with the I table. (10)
	X-14	Number of entries to be associated with the B table. (20)
	X-15	Number of entries to be associated with the F table. (20)
	X-16	Number of entries to be associated with the T table. (05)
	X-17	Number of entries to be associated with the X table. (150)
	X-18	Number of entries to be associated with the S table. (05)
	X-19	Number of entries to be associated with the Program Selection Table. (260)
X-37	X-58	Implementor-name for SELECT statement for the Card Reader.
X-45	X-73	Implementor-name for mnemonic top of page.
X-48	X-84	STANDARD or OMITTED label records clause for the printer.
X-49	X-82	Source-Computer name.
X-50	X-83	Object-Computer name.

C. PPLVP74 Messages and Error Diagnostics

The PPLVP74 messages are uniquely identified with a reference identifier. The text of each message including any other pertinent information is provided in this appendix. The reference identifier consists of a letter followed by four digits (M90l0):

Letter

M - Represents a message from the Monitor Section;
U - Represents a message from the Update Section;
X - Represents a message which could be issued from either the Update or Monitor Section.

First Digit

0 - Represents a diagnostic based on an input provided by a user. (Input to the PPLVP74 processor.) A requested action was rejected/not done due to an incorrect input parameter.

5 - Information messages; providing information in the form of a printed message.

9 - Represents an internal PPLVP74 error. This could be caused by a modification made when the PPLVP74 source program was implemented or could be a legitimate bug. When messages of this sort appear, the user should contact the FCCTS for further guidance:

Federal COBOL Compiler Testing Service
Department of the Navy
Washington, D. C. 20376
(202) 697-1247

Last Three Digits

000-999 - Represents a sequence number for each type of message.

C.1 Monitor Section Messages

M0020 NAME NOT ENTERED, SELECTION TABLE EXCEEDED

A program was requested (+ or P card; *ENVIRONMENT card referencing a + or P card) but the number of entries in the request table have all been used. The program will not be selected.

M0030 *** INVALID PPLVP74 CONTROL STATEMENT *****

The * card shown to the right of the diagnostic is not known to the PPLVP74 processor. (See 5.1.6 VP-Routine Control Statements for a list of acceptable control cards.) The card is not processed.

M0031 CKVPJCL *** DATE NOT USED, SYSTEM DATE/TIME USED.
Version 2.K and later of the PPLVP74 processor obtain the date and time of the run from the system through the ACCEPT...DATE, ACCEPT...TIME.

M0040 ***** INVALID (*LIST) OPTION - DELETED *****
M0050 ***** INVALID (*INITIALIZE) OPTION - DELETED ****

The option presented to the right of the diagnostic message is not known to the PPLVP74 processor. The option will be discarded.

M0060 ***** INVALID (*OPTION) NUMBER - DELETED *****

The *OPTnn control card presented to the right of the message has other than 1 - 27 contained in nn. The *OPTnn card will not be further processed.

M0070 ***** INVALID OPEN OPERAND *****

The name to the right of the message was the object of an *OPEN control statement and is invalid.

M0080 *** DUPLICATE (*SUBSET) ENCOUNTERED - RESULTS NOT PREDICTABLE

After processing a *SUBSET card, another was encountered. It will be processed and could lead to duplicate program requests being processed.

M0090 ***** INVALID (*SUBSET) CARD DELETED *****

The *SUBSET card presented to the right of the message is not valid. The only allowable *SUBSET cards are:

*SUBSET-A - FIPS PUB 21-1 Low Level COBOL
*SUBSET-B - FIPS PUB 21-1 Low-Intermediate COBOL
*SUBSET-C - FIPS PUB 21-1 High-Intermediate COBOL
*SUBSET-D - FIPS PUB 21-1 High Level COBOL
*SUBSET-ALL - ALL programs on the Population File

M0100 MONDRVR NO (*DATE YYMMDD) CARD FOUND - PROCESSING TERMINATED

The required *DATE card had not been found when the *END-MONITOR sentinel was processed. The PPLVP74 processor terminates itself. Provide an *DATE card and retry.

M0110 MPROCIP INVALID PPLVP74 MONITOR CONTROL CARD *****

The command shown to the right of the diagnostic message is not a proper Monitor Control card (i.e., + card, P card, - card, M card, alphabet card, * card). The card is rejected and not processed.

M0120 PROCENV ENVIRONMENTAL INFORMATION NOT FOUND *****

The name of a requested environmental file, presented to the right of the diagnostic, was not found in the Population File. The request therefore is not processed.

M5010 MONDRVR ** NO (*END-MONITOR) FOUND END-OF-FILE - PROCESSING CONTINUES WITH/NO UPDATES

The *END-MONITOR which indicates the end of the Monitor Section was not found. Instead an end-of-file was encountered. If an Update Section was present, it wasn't found and therefore not processed.

M5020 COPIED ENVIRONMENT PARAMETER

The PPLVP74 input parameter printed to the right of COPIED text was right of it was copied from an environmental entry on the Population File. The ENVIRONMENT option of the *LIST command must preceed the *ENVIRONMENT card in order for the contents of the environmental entry to be listed as each command is processed.

M5021 COPIED XXXXXXXX NNN RECORDS

The environmental entry 'XXXXXXXXXXX' has been extracted from the Population File. 'NNN' records were contained in that entry. (If the ENVIRONMENT option of the *LIST card was not present, then the entry would not have been listed as each statement was processed.

M9010 INVALID PPLVP74 MONITOR CONTROL CARD

PPLVP74 internal error. A +(P) card was present in an *INSERT-C request (which is not permitted) and an attempt to process it triggered this message. The +(P) card should be placed in the Monitor Section in order to be accepted.

M9020 OPCTFIL - OPEN TO CTL FILE NOT PREVIOUSLY CLOSED

An internal OPEN was issued to the control card file which was already in an open status.

C.2 Update Section Messages

U0010 INSCTL ILLEGAL (*INSERT-C) COMMAND

A PPLVP74 Monitor Command was detected in an *INSERT-C packet in the Update Section which is not among the following:

*OPTION
*PROGID
*LIST
*BCD-P
*BCD-C
Alphabet cards (Except for 'I')

The command is not processed.

U0020 NO (*HEADER) DATA BYPASSED

Within an *INSERT-P (insert program) a data card was encountered which was not between a set of *HEADER/*END-OF cards. This can be caused by a missing *HEADER card, source cards preceeding the *HEADER card, etc. The images which have been detected as being out of place are bypassed in search of a *HEADER card or an *END-INSERT card.

U0030 INVALID (*INSERT) TYPE; REGULAR ASSUMED

The -n suffixed to the *INSERT card is not recognized and the insert packet is processed as though there had been no suffix. -C, -P, -D represent Command, Programs, and Data respectively.

U0040 NO (*END-INSERT), FOUND END-OF-FILE

The end-of-file was encountered in the Control Card File while the PPLVP74 was expecting an *END-INSERT to terminate the previous insert packet. The insertion is terminated as though the *END-INSERT had been present. This also terminates the Update Section as though an *END-UPDATE card had been processed.

U0041 NO (*END-INSERT) - PROCESSING CONTINUES WITH *****

An insert packet was terminated but not by the presence of an *END-INSERT statement. Processing will continue with the Update Command listed to the right of the diagnostic message.

U0050 INVALID UPDATE HEADER (*START, *INSERT, *END-UPDATE)

One of the three Update Commands was expcted but not found. The insert file will be searched until one of the three are located or until the end-of-file is located. The bypassed data cards are not processed.

U0060 UPDTRTN UPDATE ABORTED IN GOOD TASTE, FORTRAN NOT SUPPORTED

Updates were indicated for a program other than COBOL. The

Update Section is bypassed since only COBOL programs can be updated as of Version 1.16 PPLVP74.

U0070 VRFYUPDT ILLEGAL *BEGIN-UPDATE

The option supplied on the *BEGIN-UPDATE card was not recognized. A temporary update is assumed.

U0071 VRFYUPDT (*BEGIN-UPDATE) NOT FOUND - UPDATE CANCELLED

The *BEGIN-UPDATE header which must follow the *END-MONITOR card to initiate the Update Section was not found. The update is cancelled and any updates present are ignored ad bypassed.

U0080 XCARD UNDEFINED XXXXYNNN CARD, NOTHING DONE *****

The XXXXYNNNX card shown to the right of the diagnostic had not been previously defined through an X-card in the Monitor Section or in *INSERT-C packet request in the Update Section. The image will be unchanged as it is written to the output file.

U5010 UPDATES BYPASSED (PROGRAM WAS NOT SELECTED) *****

Updates were encountered for the program named to the right of the message, but because the program had not been selected, the updates are bypassed and the next Update Section header located. This message is for information only.

U5020 GENJCL DATA GENERATED NNNNNNN RECORDS

The number of records indicated were generated for the program being processed. This is triggered by the 'DATA****' indicator being present as one of the Ending Control cards. (E-card)

U5030 SKPUPDTS * PROGRAM DELETED ***** XXXX

During a permanent update (creation of a new Population File), the program named to the right of the messages was not selected (+ or P card) and is logically deleted from the new Population File.

(+ or P card) and is logically deleted

U5031 SKUPDATES *UPDATES BYPASSED XXXXX

The updates for a program not selected during a permanent update (creation of a new Population File) are bypassed. The next set of updates/insertion is located.

U5040 UPDTRTN SEQUENCE ERROR

An update card for a program has a sequence number which is less than the present update card. The alleged error card will be inserted immediately following the prior update card. The cards should be placed in the proper order and the update rerun.

U5050 PROGRAM NOT FOUND

The program named to the left of the diagnostic was selected (+ or P card) but not found in the Population File. A duplicate request for the same program will cause (1) the program to be selected and (2) the above message indicating that the program was not found. If the Population File (SELECT OPTIONAL...) is not present then none of the programs will be found.

U5060 UPDT001 POPULATION FILE NOT PRESENT, FILE CREATION ASSUMED

The PPLVP74 processor attempted to open the Population File to obtain the tape creation date and the owner information and the file was not present. It assumes that all programs to be manipulated will be input from the control card file.

U9010 * CLOSE ATTEMPTED ON CLOSED FILE (CONTROL-CARD-FILE)

A second internal request to close the Control card file has been intercepted and refused.

U9020 * CLOSE ATTEMPTED ON CLOSED FILE (POP-FILE)
A second internal request to close the Population File has been intercepted and refused.

U9030 * CLOSE ATTEMPTED ON CLOSED FILE (PRINT-FILE)

A second internal request to close the Print file has been intercepted and refused. This message is issued via the DISPLAY statement since the Printer file is not in open status.

U9040 * CLOSE ATTEMPTED ON CLOSED FILE (UP-POP)

A second internal request to close the output file containing the selected programs and JCL has been intercepted and refused.

U9050 * CLOSE ATTEMPTED ON CLOSED FILE (UP-POP)

A second internal request to close the updated Population File was intercepted and refused.

U9060 * CLOSE ATTEMPTED ON CLOSED FILE (VP-WORK-1)

An attempt to close VP-WORK1 which was not in an OPEN status was refused.

U9070 GENJCL INVALID INTERNAL CTL NUMBER - SYSTEM ERROR

U9071 GENJCL INVALID INTERNAL CTL NUMBER - SYSTEM ERROR

Internally, a invalid number has been passed to the JCL generator subroutine.

U9080 OPPOPFIL - OPEN TO POP-FILE NOT PREVIOUSLY CLOSED

An internal request to open the Population File, when it was already

in an open state, was intercepted and refused.

U9090 OPPRTFIL - OPEN TO PRINT-FILE NOT PREVIOUSLY CLOSED

An internal request to open the Print File, when it was already in an open state, was intercepted and refused.

U9100 OPSORFIL - OPEN TO SOURCE FILE NOT PREVIOUSLY CLOSED

An internal request to open the Source File, when it was already in an open state, was intercepted and refused.

U9110 OPUPPFIL - OPEN TO UP-POP FILE NOT PREVIOUSLY CLOSED

An internal request to open the updated Population File, when it was already in an open state, was intercepted and refused.

U9120 OPWRKFIL - OPEN TO VPWORK1 NOT PREVIOUSLY CLOSED

An internal request to open VP-WORK1 file, when it was already in a open state, was intercepted and refused.

U9122 WTWK01 - ATTEMPTED TO WRITE CLOSED WORK FILE

U9130 WTSORJCL ATTEMPTED TO WRITE CLOSED SOURCE FILE

During a permanent update (in which the Source File is not normally open) an attempt to write generated JCL to the Source file was attempted.

U9132 WTSORSOR ATTEMPT TO WRITE CLOSED SOURCE FILE

An internal attempt was made to issue an output command to the Source File which was closed at the time.

U9140 WTUPPFIL ATTEMPT TO WRITE CLOSED UP-POP FILE

An internal attempt was made to issue an output command to the updated Population File which was closed at the time.

U9150 XCARD PPLVP74 SYSTEM ERROR

An internal problem has arisen within the module which handles the X card replacement in a source program. This is probably caused by processing a source program containing an XXX or ZZZ statement which exceeds the maximum allowable.

C.3 Other Messages

X0010 *** OPERAND GREATER 18 CHARACTERS, PROCESSING TERMINATED

The subroutine which processes input patameters has encountered an operand greater than 18 digits. This is an error since there are no operands acceptable to the PPLVP74 processor which are that long. The card will not be further processed. Any options accepted up to but not including the erroneous operand will be accepted. Anything beyond that operand will not be processed.

X0020 *** MORE THAN 10 OPERANDS ENCOUNTERED ... DELETED

An eleventh field has been detected in a PPLVP74 parameter card (* card). The eleventh field and anything beyond that field are not processed.

X0030 LDALPHCD INVALID ID PLACEMENT (01-75)

The ID placement (columns 5-6) in the previously listed PPLVP74 alphabet card is in error. It is shown to the right of the diagnostic message. The range is from 01 to 75. The ID placement will not take place. The card should be corrected and processed again.

X0040 LDALPHCD INVALID ALPHABET CARD NUMBER XX

X0041 LDALPHCD INVALID (A) CARD NUMBER

X0042 LDALPHCD INVALID ALPHABET CARD NUMBER XX

The alphabet card number (column 2-4) is not within the range for that card. This number is set at PPLVP generation time. The invalid number is shown to the right of the diagnostic message. If the two positions are blank or contain other than numeric data then they will be rejected. If a number that is larger is required, then the PPLVP74 processor should be regenerated specifying a larger number for that alphabet card.

X0050 LDALPCH INVALID (A) CARD TYPE

The 'A' card type identifier (column 2) is not recognized. -, P, S, D are the four valid characters acceptable in Column 2. (An exception would be an integer if the number of 'A' cards specified at PPLVP74 generation time was greater than 99.)

X5010 RDCTLFIL EOF ENCOUNTERED ON CTL-FILE

An end of file condition was encountered while attempting to access the next record within the Control Card File.

X9010 RDCTLFIL ATTEMPTED TO READ CLOSED CTL-FILE

Internally an attempt was issued to read the Control Card file but it is not in an open status.

X9020 RDPOPFIL ATTEMPT TO READ CLOSED POP-FILE

Internally an attempt was made to read the Population File which was not in an open status.

X9030 WTUDNOTE ATTEMPT TO WRITE CLOSE PRINT-FILE

An internal request to write the Printer File is denied because the file is not opened.

APPENDIX D: SUMMARY OF CCVS PROGRAMS

Appendix D briefly describes the features tested in each audit program of the 1974 COBOL Compiler Validation System version 3.0. The X-numbers used by each program are listed. The X-numbers in the basic set are used by each audit program except for a few Library and Inter-program Communication programs. The X-numbers in the basic set are 55, 82, 83, and 84.

D.1 COMMUNICATION MODULE

CM431

X-Numbers: Basic set plus 30, 31, 32, 33, 35.

Features Tested: This program is a FIPS flagging test and contains Communication Module elements from both ANSI levels 1 and 2. Both ANSI levels belong in Federal Standard COBOL High-Intermediate and High levels. The program does not required a Communication environment to execute. Should execution of the Communication statements be desired, the WORKING-STORAGE item, COMP-FLAG should be modified to a non-zero VALUE before compilation of CM431. The following statements should be flagged whenever monitoring at the Low or Low-intermediate levels:

COMMUNICATION SECTION (and all subordinate entries)
ACCEPT MESSAGE COUNT
ENABLE cd-name
DISABLE cd-name
RECEIVE cd-name
SEND cd-name

D.2 DEBUG MODULE

DB101

X-Numbers: Basic set only.

Features Tested: DB101 tests the basic operation of the Debug Module when both the compile and object time debugging switches are turned on. The program contains both debug lines and simple debugging procedures. The debugging procedures are specified for procedure-names and procedure-name series. The following conditions are evaluated for the 'DEBUG-ITEM' register:

(1) Start of program
(2) Reference by 'ALTER'
(3) Reference by 'GO TO'
(4) Reference by 'PERFORM'
(5) Sequential passage of control (fall through)

Before beginning execution of the object program, the job control language commands necessary to activate (turn on) the object time debugging switch should be submitted.

DB102

X-Numbers: Basic set only.

Features Tested: DB102 tests the basic operation of the Debug Module facilities when the compile time debugging switch is on and the object time switch is off. All debug lines and debugging procedures should be included in compilation and generate code.

Before beginning execution of the object program, the job control language commands necessary to deactivate (turn off) the object time debugging switch must be submitted.

At execution time, code generated from debug lines should be executed, but debugging procedures should be deactivated by the object time switch.

DB103

X-Numbers: Basic set only.

Features Tested: DB103 tests the basic operation of Debug Module facilities when the compile time debugging switch is off. All debug lines should be treated as comments and no code should be generated for either debug lines or debugging procedures.

The object program for DB103 should be executed twice: once with the object time debugging switch enabled (on), and once with the object time debugging switch disabled (off). Both execution runs should yield the same results as the setting of the object time debugging switch should make no difference since the compile time debugging switch was initially disabled.

DB104

X-Numbers: Basic set plus 14, 27, 69, 74, 75.

Features Tested: DB104 tests the capability of the Debug Module to handle procedures tied to sort input, sort output and file declarative procedures. This program is to be compiled and executed with both compile and object time debugging switches enabled. The program first builds a sequential file containing 99 eighty character records. This file is then sorted.

All debugging procedures should be included in compilation and generate code. Before beginning execution of the object program, the job control language commands necessary to activate the object time debugging switch must be submitted.

Execution of the program's sort should trigger debugging procedures at the beginning of the sort input and sort output procedures. During execution of the sort input procedure, an end-of-file condition on the input file should trigger a declarative procedure associated with the file, and this in turn should cause execution of a debugging procedure monitoring the file declarative procedure.

The performance of the SORT verb is not checked in DB104.

DB105

X-Numbers: Basic set only.

Features Tested: DB105 tests the capability of the Debug Module to monitor all procedures with a single debugging declarative. This program is to be compiled and executed with both compile and object time debugging switches on. The debugging procedure should be included in the compilation and code should be generated for the procedure. During execution, each procedure should trigger the debugging procedure which stacks the name of the procedure calling it. Prior to being stacked, each name is potentially adjusted by modifying a fixed-location numeric subfield in the name. If the program executes properly, the names that are stacked will be unique and in an ascending sequence in the numeric subfield. Near the end of the program the stacking function is disabled, and the name stack is compared to a static table containing procedure-names in the order in which the procedures should

have stacked.

DB105's output report differs slightly from the normal CCVS format. If execution is perfect, the report will consist of 227 lines showing:

(1) Program procedure name as it appears in the program,
(2) Adjusted procedure name after its numeric subfield has been adjusted,
(3) Adjusted debug-name that was stacked by the debugging procedure.

Nominally, the numeric subfields of the procedure names should appear in ascending sequence. Any deviations in the stacking sequence from the expected sequence will cause additional report lines to be generated with one or more columns blank. If nothing ever appears in the 'ADJUSTED DEBUG-NAME' column, it may be assumed that the debugging procedure was never executed.

It is a fundamental assumption of DB105 that when a section is entered, the debugging section will be called twice, once for the section name and once for the paragraph-name that immediately follows the section-name. Additionally, DB105 traps any failures in program flow caused by a failure of verbs from the nucleus module. These failures are summed and reported at the bottom of DB105's report. If any procedure-names beginning with 'PROC-000' appear in the 'ADJUSTED DEBUG-NAME' column of the report, these result from execution of procedures which should not have been executed if the program had followed the proper control flow sequence.

DB201

X-Numbers: Basic set only.

Features Tested: DB201 tests the capability of the Debug Module to handle debugging procedures which are monitoring identifiers specified with and without the 'ALL REFERENCES' option. This program is to be compiled and executed with both compile and object time debugging switches enabled. The debugging procedures should be included in compilation and code should be generated for them. Debugging sections on the following conditions are analyzed:

(1) Reference to identifier within 'VARYING', 'AFTER' and 'UNTIL' phrases of PERFORM statements,
(2) Reference to changed and unchanged identifier fields,
(3) Reference to subscripted identifiers,
(4) Reference to qualified identifiers,
(5) Reference to identifier used in GO TO DEPENDING,
(6) Reference to identifier in unexecuted statements,

(7) Multiple references to same identifier in same statements.

DB202

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: DB202 tests the capability of the Debug Module to handle debugging procedures which are monitoring I-O functions of the Sequential I-O Module. This program is to be compiled and executed with both compile and object time debugging switches on. The debugging procedures should be included in compilation and code should be generated for them.

During execution a sequential file is created containing 80-character records. The file is then read. Execution of OPEN, READ, and WRITE statements should trigger the appropriate debugging procedures.

DB203

X-Numbers: Basic set plus 24, 44, 56, 69, 74, 75.

Features Tested: DB203 tests the capability of the Debug Module to handle debugging procedures which are monitoring I-O functions of the Relative I-O or Indexed I-O modules. This program is to be compiled and executed with both compile and object time debugging switches on. The debugging procedures should be included in compilation and code should be generated for them. During execution a file is assigned in dynamic mode, created sequentially and accessed both sequentially and randomly. Its records are 80-characters in length. Execution of OPEN, READ, WRITE, REWRITE, START, and DELETE statements should trigger the appropriate debugging procedures.

DB204

X-Numbers: Basic set plus 14, 15, 16, 27, 69, 74, 75, 76, 77.

Features Tested: DB204 tests the capability of the Debug Module to handle a debugging procedure which is monitoring a MERGE OUTPUT procedure. This program is to be compiled and executed with both compile and object time debugging switches on. The debugging procedure should be included in compilation and code should be generated for the procedure. During execution, two sequential files are created with each containing 80-character records in sorted order. The two files are then merged. Execution of the merge operation should trigger the debugging procedure linked to the merge output procedure name.

DB421

X-Numbers: Basic set only.

Features Tested: The program DB421 contains level 1 Debug language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The DEB level 1 elements in this program are contained in the Low-Intermediate level of Federal Standard COBOL. The Environment division clause to be flagged is

```
WITH DEBUGGING MODE.
```

The Procedure Division statements to be flagged are

```
USE FOR DEBUGGING ON procedure-name.
USE FOR DEBUGGING ON procedure-name-series.
Debugging Lines.
All statements containing references to the DEBUG-ITEM
register and to its component sub-registers

    DEBUG-LINE
    DEBUG-NAME
    DEBUG-CONTENTS.
```

DB422

X-Numbers: Basic set only.

Features Tested: The program DB422 contains level 1 Debug language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The DEB level 1 elements in this program are contained in the Low-Intermediate level of Federal Standard COBOL. The Environment Division clause to be flagged is

```
WITH DEBUGGING MODE.
```

The Procedure Division statement to be flagged is

```
USE FOR DEBUGGING ON ALL PROCEDURES.
```

DB431

X-Numbers: Basic set plus 14, 30, 32, 69, 74, 75.

Features Tested: The program DB431 contains level 2 Debug language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The DEB level 2 elements in this program are contained in the High-Intermediate level of Federal Standard COBOL. The Procedure Division statements to be flagged are

```
USE FOR DEBUGGING ON cd-name.
USE FOR DEBUGGING ON filename-series.
USE FOR DEBUGGING ON ALL REFERENCES OF identifier.
```

In order to produce a syntactically correct program, DB431 also contains elements from level 1 of the Sequential I-O and Communication Modules; these statements will not be executed unless the WORKING-STORAGE item COMP-FLAG is modified to a non-zero VALUE prior to compilation. This obviates the necessity to set up a Communication environment in order to run DB431.

D.3 INTER-PROGRAM COMMUNICATION MODULE

IC101

X-Numbers: Basic set only.

Features Tested: This routine checks the use of the CALL statement with one parameter in the USING phrase. Subsequent calls check that the called routine remains in the last used state. This program calls subprogram IC102.

There are no test deletion paragraphs in this routine since these are the basic CALL tests and if a CALL statement is rejected there is no reason to run the routine.

The first three CALLs use a data-name the same as the name in the subprogram PROCEDURE DIVISION header USING phrase. The last two CALLs use a data-name different from the name in the subprogram. The PICTURE clauses for data-names in the USING phrases of the called and calling programs are identical.

IC102

X-Numbers: Basic set only.

Features Tested: This subprogram is called by IC101 and tests the use of the LINKAGE SECTION and the USING phrase in the PROCEDURE DIVISION header.

IC103

X-Numbers: Basic set only.

Features Tested: This program tests the use of multiple data-names in the USING phrase of the CALL statement. Two 01 group items and an elementary 77 item are the parameters. The data definitions for the group item parameters are not the same as in the subprogram but the number of characters are identical. This program CALLs subprograms IC104 and IC105.

IC104

X-Numbers: Basic set only.

Features Tested: The subprogram IC104 has three operands in the USING phrase of the PROCEDURE DIVISION header. Two operands are 01 group items and the third is an elementary 77 item. The data descriptions of these operands in the LINKAGE SECTION are not the same as the data descriptions in the WORKING-STORAGE SECTION of the calling program, but an equal number of char-

acter positions are defined. The calling program is IC103.

IC105

X-Numbers: Basic set only.

Features Tested: The subprogram IC105 has two operands in the PROCEDURE DIVISION header and the routine contains four EXIT PROGRAM statements. The calling program is IC103.

IC106

X-Numbers: Basic set only.

Features Tested: This program CALLs subprogram IC107 with two tables and an index data item referenced in the USING phrase of the CALL statement. Both of the tables contain an INDEXED BY clause.

The tests in this program verify that:

(1) The indices in the main program and the subprogram are separate.
(2) An index data item set in a main program can be used to set an index in a subprogram.
(3) Tables can be shared between a main program and a subprogram.

IC107

X-Numbers: Basic set only.

Features Tested: The subprogram IC107 contains tables and an index data item which are defined in the LINKAGE SECTION and named as operands in the USING phrase of the PROCEDURE DIVISION header. One of the tables has an index defined for it. This index should be separate from the index defined for the same table in the main program IC106, but no space should be allocated for the tables defined in the LINKAGE SECTION. The index data item is set in the main program prior to calling IC107, and it is used in this subprogram to set an index for referencing the table in the subprogram.

IC108

X-Numbers: Basic set only.

Features Tested: The program IC108 is the main program which starts a sequence of CALLs to the subprograms IC109, IC110 and IC111. Parameters are set in each of these subprograms and checked when control is returned to the main program.

IC109

X-Numbers: Basic set only.

Features Tested: The subprogram IC109 is the first subprogram in a sequence of CALLs which start in the main program IC108. IC109 CALLs IC110 with one operand in the WORKING-STORAGE SECTION and one operand in the LINKAGE SECTION.

IC110

X-Numbers: Basic set only.

Features Tested: The subprogram IC110 is the second subprogram in a sequence of CALLs which start in the main program IC108. This subprogram CALLs IC111 with operands in the LINKAGE SECTION and in the WORKING-STORAGE SECTION. The subprogram IC110 is CALLed by IC109.

IC111

X-Numbers: Basic set only.

Features Tested: The subprogram IC111 is the last subprogram CALLed in a sequence of subprogram CALLs which is started in the main program IC108. The subprogram IC111 is CALLed by the subprogram IC110.

IC112

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: The routine IC112 is a main program which has a file description for a sequential mass storage file with fixed-length records. The file is created, CLOSEd and OPENed as an input file. The main program reads the file and verifies that the file is correct. The file is CLOSEd and OPENed again as an input file. A record is read and a CALL is made to the subprogram IC113 with the file description 01 record listed as one of the operands of the USING phrase. The subprogram IC113 compares the fields in the input record to the values written when the file was created.

This program was adapted from the Sequential I-O tests contained in program SQ104. If any errors occur in running the routine SQ104 the results of the tests in routines IC112 and IC113 are inconclusive.

IC113

X-Numbers: Basic set only.

Features Tested: The subprogram IC113 is CALLed by the main program IC112 which references a file description record in the USING phrase of the CALL statement. IC113 checks the values in the file record described in the LINKAGE SECTION of the subprogram. If any errors are encountered, the error flag is set to 1 and the RECORDS-IN-ERROR counter is incremented by 1.

IC114

X-Numbers: Basic set only.

Features Tested: The program IC114 is a main program which CALLs the subprogram IC115. The purpose of these programs is to verify that a FILE SECTION, WORKING-STORAGE SECTION and a LINKAGE SECTION can appear in a subprogram, and that a file can be written and read within a subprogram.

The program IC114 CALLs IC115 to create and verify the file. Subsequent CALLs to the subprogram are made to read the file and return a record to the main program which checks the record contents.

The subprogram IC115 is adapted from the Sequential I-O program SQ104. If SQ104 does not execute correctly then the results of these tests are inconclusive.

IC115

X-Numbers: Basic set only.

Features Tested: The program IC115 is a subprogram CALLed by IC114. This subprogram contains a FILE SECTION, WORKING-STORAGE SECTION and LINKAGE SECTION. A file is created and verified in this routine. The file is opened and read again. Each record is checked by moving it to the LINKAGE SECTION and returning to the main program to verify the record contents. The printing of the output report for the test results is performed by returning to the main program.

This subprogram is adapted from the Sequential I-O routine SQ104. If that routine does not perform correctly then the results of these tests are inconclusive.

IC116

X-Numbers: Basic set only.

Features Tested: The program IC116 tests the CALL statement without the optional USING phrase. The subprograms IC117 and IC118 test the omission of the optional USING phrase in the

Procedure Division header of a subprogram. The program IC116 calls the subprogram IC117 which in turn calls the subprogram IC118. The subprograms contain DISPLAY statements which show the execution sequence.

IC117

X-Numbers: Basic set only.

Features Tested: The subprogram IC117 does not contain a Linkage Section or the optional USING phrase in the Procedure Division header. IC117 is called by the main program IC116 and calls IC118. The CALL statement also does not have the optional USING phrase. DISPLAY statements are used to verify the program execution sequence.

IC118

X-Numbers: Basic set only.

Features Tested: The subprogram IC118 is called by the subprogram IC117. The subprogram IC118 does not contain a Linkage Section or the optional USING phrase in the Procedure Division header. A DISPLAY statement is executed to verify that this subprogram was executed.

IC119

X-Numbers: Basic set only.

Features Tested: The program IC119 is a main program with a Linkage Section and a Procedure Divison header with the optional USING phrase. The purpose of this program is to test that a program can function as either a main program or as a subprogram depending upon whether it is called or not. The procedure division statements which are executed do not reference data items in the Linkage Section, but the statements which are not executed do reference data items defined in the Linkage Section.

IC120

X-Numbers: Basic set only.

Features Tested: The subprogram IC120 contains a Linkage Section defining five records. The Procedure Division header for this subprogram names only two of the five records contained in the Linkage Section. The main program IC119 calls this subprogram.

IC121

X-Numbers: Basic set plus 27.

Features Tested: The main program IC121 calls the subprogram IC122 which contains a SORT statement. The purpose of these programs is to verify that a SORT statement functions correctly in a subprogram. Control is not returned to this program since IC122 contains a STOP RUN statement.

IC122

X-Numbers: Basic set plus 27.

Features Tested: The subprogram IC122 tests the use of a SORT statement in a segmented subprogram. The first non-declarative section of the subprogram consists of a SORT statement and a STOP RUN statement in a fixed permanent segment. The sort input procedure and sort output procedure are contained in two independent segments. The main program IC121 calls this subprogram and the subprogram IC123 is called from the output procedure section to print the output report.

IC123

X-Numbers: Basic set plus 27.

Features Tested: The subprogram IC123 prints the test results for the program set IC121, IC122 and IC123. It is called by the main program IC121 and the subprogram IC122. A linkage variable indicates whether the heading, footing or a report line is to be printed.

IC151

X-Numbers: Basic set only.

Features Tested: This routine checks the use of the CALL statement with one parameter in the USING phrase. Subsequent CALLs check that the CALLed routine remains in the last used state. This program CALLs subprogram IC152.

There are no delete paragraphs in this routine since these are the basic CALL tests and if a CALL statement is rejected there is no reason to run the routine.

The first three CALLs use a data-name the same as the name in the subprogram PROCEDURE DIVISION header USING phrase. The last two CALLs use a data-name different from the name in the subprogram. The PIC clauses for data-names in the USING phrases of the CALLed and CALLing programs are identical.

IC152

X-Numbers: Basic set only.

Features Tested: This subprogram is CALLed by IC151 and tests the use of the LINKAGE SECTION and USING phrase in the PROCEDURE DIVISION header.

IC201

X-Numbers: Basic set only.

Features Tested: The program IC201 CALLs subprogram IC202 and tests the CALL statement with an identifier as an operand, and four operands in the USING phrase. The repetition of a data-name in the USING phrase is tested, and the use of the ON OVERFLOW phrase in a CALL statement is syntactically checked in the program.

IC202

X-Numbers: Basic set only.

Features Tested: The subprogram IC202 is CALLed by program IC201. The subprogram has four operands in the USING phrase of the PROCEDURE DIVISION header.

IC203

X-Numbers: Basic set only.

Features Tested: The program IC203 tests the use of the CANCEL statement. This program CALLs IC204 and IC205 and verifies that the initial CALL to a subprogram and the first CALL after a CANCEL results in a subprogram being entered in its initial state. The program also CANCELs a program which has not been CALLed, in which case control should pass to the next sentence.

IC204

X-Numbers: Basic set only.

Features Tested: The subprogram IC204 has two variables in WORKING-STORAGE which are initialized by a VALUE statement. The data contents of these variables are modified during execution of the subprogram. Indicators are set for variables in the LINKAGE SECTION which relate how many times the subprogram has been CALLed since it was initialized, and whether or not the subprogram is in its initial state.

IC205

X-Numbers: Basic set only.

Features Tested: The subprogram IC205 tests the use of the CANCEL statement within a subprogram. This subprogram is CALLed by IC203 and CALLs subprograms IC204 and IC206.

IC206

X-Numbers: Basic set only.

Features Tested: The subprogram IC206 is CALLed by the subprogram IC205. The subprogram is then CANCELed and CALLed again. The program IC205 checks if IC206 was in its initial state on the first CALL after the program was CANCELed.

The LINKAGE parameter DN1 contains the number of times IC206 has been CALLed since initialization.

IC207

X-Numbers: Basic set only.

Features Tested: The program IC207 defines a variable length table. The table and the variable containing the table length are operands in a CALL statement USING phrase. Also an index is defined for the table and an index data item is used to pass an index value for a table reference to and from the subprogram IC208.

IC208

X-Numbers: Basic set only.

Features Tested: The subprogram IC208 contains tables and an index data item which are defined in the LINKAGE SECTION and named as operands in the USING phrase of the PROCEDURE DIVISION header. One of the tables is defined with an OCCURS DEPENDING ON clause and has condition-name entries associated with it. The SEARCH statement is used to test a variable length table capability.

IC421

X-Numbers: Basic set only.

Features Tested: The program IC421 contains the level 1 Inter-Program Communication language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The IPC level 1 elements in this program are contained in the Low-Intermediate Level of Federal Standard COBOL. The Procedure Divison statement to be flagged is

```
CALL literal USING data-name series.
```

The subprogram IC422 is called by this program.

IC422

X-Numbers: Basic set only.

Features Tested: The subprogram IC422 contains the level 1 Inter-Program Communication language features for subprograms for purposes of testing the FIPS-PUB 21-1 flagging requirement. The IPC level 1 elements in this subprogram are contained in the Low-Intermediate Level of Federal Standard COBOL. The Data Division entry to be flagged is

```
LINKAGE SECTION.
```

The Procedure Division statements to be flagged are

```
PROCEDURE DIVISION USING data-name series.
EXIT PROGRAM.
```

The main program IC421 calls the subprogram IC422.

IC431

X-Numbers: Basic set only.

Features Tested: The program IC431 contains the level 2 Inter-Program Communication language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The IPC level 2 elements in this program are contained in the High-Intermediate Level of Federal Standard COBOL. The Procedure Division statements to be flagged are

```
CALL identifier USING data-name series.
CALL literal USING data-name series
   ON OVERFLOW imperative-statement.
CALL identifier USING data-name series
   ON OVERFLOW imperative-statement.
CANCEL literal.
CANCEL identifier.
```

The subprogram IC432 is called by the program IC431.

IC432

X-Numbers: Basic set only.

Features Tested: The subprogram IC432 is called by the main program IC431. Since there are no level 2 Inter-Program Communication

elements for subprograms, this subprogram contains the IPC level 1 elements for purposes of testing the FIPS PUB 21-1 flagging requirement. The IPC level 1 elements in this subprogram are contained in the Low-Intermediate Level of Federal Standard COBOL. The Data Divison entry to be flagged is

```
LINKAGE SECTION.
```

The Procedure Division statements to be flagged are

```
PROCEDURE DIVISON USING data-name series.
EXIT PROGRAM.
```

D.4 INDEXED I-O MODULE

IX101

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length indexed file whose access mode is sequential. The indexed file created in IX101 is passed to IX102 for further processing.

IX102

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program is used to verify the indexed file created in IX101. The access mode for the file in this program is random. The file is verified as to the existance and accuracy of the 500 records created by IX101. Then certain records are selectively updated, and the accuracy of each record is verified again. The updated indexed file is passed to IX103 for further processing.

IX103

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program is used to verify the indexed file updated in IX102. The access mode for the file in this program is sequential. The file is verified for accuracy of its 500 records. Then certain records of the file are selectively deleted, and the accuracy of each record is verified again.

IX104

X-Numbers: Basic set plus 25, 45, 69, 74, 75.

Features Tested: This program is used to create an indexed file whose access mode is sequential and then selectively update the file. The FILE STATUS data item is tested for each OPEN, CLOSE, READ, and REWRITE statement. For this program the READ, WRITE, and REWRITE statements are used without the appropriate AT END and INVALID KEY phrases.

IX105

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length indexed file whose access mode is sequential. The indexed file created in IX105 is passed to IX106 for further processing.

IX106

X-Numbers: Basic set plus 24, 44, 45, 51, 53, 69, 74, 75.

Features Tested: This program tests the use of the RERUN clause with an indexed organized file whose access mode is sequential. IX106 processes the file created in IX105.

Special Considerations:

This program requires that at least one of the permissible RERUN clauses for indexed files be specified for the X-53 card.

IX107

X-Numbers: Basic set plus 24, 25, 44, 45, 69, 74, 75, 76.

Features Tested: This program tests the use of the SAME AREA clause for two indexed files whose access mode is sequential. After each file is created they are processed and verified using various combinations of the READ statement with the AT END phrase.

IX201

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length indexed file whose access mode is sequential. The indexed file created in IX201 is passed to IX202 for further processing.

IX202

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program is used to verify the indexed file created in IX201. The ACCESS MODE IS DYNAMIC clause is used for the file in this program. The file is verified as to the existance and accuracy of the 500 records created by IX201. Then certain records are selectively updated, and the accuracy of each record is verified again. The updated indexed file is passed to IX203 for further processing.

IX203

X-Numbers: Basic set plus 24, 44, 69, 74, 75.

Features Tested: This program is the third in a series. It is used to verify the indexed file updated in IX202 using the ACCESS MODE IS DYNAMIC clause. The file is verified sequentially for accuracy of its 500 records. Then certain records of the file are selectively deleted, and the accuracy of each record is verified again.

IX204

X-Numbers: Basic set plus 25, 45, 69, 74, 75.

Features Tested: This program is used to create an indexed file sequentially using the ACCESS DYNAMIC clause and then update selective records within the file using the REWRITE statement. The FILE STATUS data item is tested for each OPEN, CLOSE, READ and REWRITE statement. For this program the READ, WRITE, and REWRITE statements are used without the optional AT END or INVALID KEY phrases.

IX205

X-Numbers: Basic set plus 24, 25, 44, 45, 69, 74, 75, 86.

Features Tested: This program is used to test various permissible syntactical constructs of level 2 of the Indexed I-O module. The constructs tested are:

ACCESS MODE DYNAMIC
ALTERNATE RECORD KEY without the duplicates options
RESERVE clause
SAME clause
BLOCK CONTAINS...TO... clause
VALUE OF... clause

IX206

X-Numbers: Basic set plus 24, 25, 44, 45, 69, 74, 75, 86, 87, 88.

Features Tested: This program is used to test various permissible syntactical constructs of level 2 of the Indexed I-O module. One file is created and accessed using the ACCESS DYNAMIC clause. A second accessed using the ACCESS MODE IS SEQUENTIAL clause. The constructs tested are:

ACCESS DYNAMIC
ACCESS MODE IS SEQUENTIAL
ALTERNATE RECORD KEY without the duplicates options
RESERVE clause
SAME clause

BLOCK CONTAINS...TO... clause
VALUE OF implementor-name series

IX207

X-Numbers: Basic set plus 24, 25, 44, 45, 69, 74, 75, 76.

Features Tested: This program is used to test various permissible syntactical contructs of level 2 of the Indexed I-O module. A file is created and accessed in the SEQUENTIAL ACCESS mode. The constructs tested are:

Ordering of clauses in file control entry
ALTERNATE RECORD KEY with duplicates options
USE AFTER STANDARD EXCEPTION file-name series
FILE STATUS clause

IX208

X-Numbers: Basic set plus 24, 25, 44, 45, 69, 74, 75, 76.

Features Tested: This program is used to test various permissible syntactical constructs of level 2 of the Indexed I-O module. Two indexed files are created and verified. One file is created with ACCESS MODE IS DYNAMIC while the other file is created with ACCESS MODE IS SEQUENTIAL. After each file has been verified, it is read sequentially using the START statement to logically position the file.

IX209

X-Numbers: Basic set plus 24, 44, 69, 74 and 75.

Features Tested: IX209 tests use of the START --- EQUAL TO --- statement using the prime record key and each of the alternate record keys as the key of reference. Included within the START statement is either the data name specified in the key clause or a data item that is subordinate to the key name. Different key values are used for testing. If a key value is provided which matches a record in the file when the START statement is executed, then the record is expected to be made available by the subsequent READ statement. If a key value is provided which does not match any record in the file, then the INVALID KEY path is expected to be taken. The contents of the FILE STATUS key are saved and checked after the execution of each START statement.

IX210

X-Numbers: Basic set plus 24, 44, 69, 74, and 75.

Features Tested: IX210 tests use of the START --- GREATER THAN --- statement

using the prime record key and an alternate record key as the key of reference. Included within the START statement is either the data name specified in the key clause or a data item that is subordinate to the key name. Different key values are used for testing. If a key value is provided which matches a record in the file when the START statement is executed, then the record is expected to be made available by the subsequent READ statement. If a key value is provided which does not match any record in the file, then the INVALID KEY path is expected to be taken. The contents of the FILE STATUS key are saved and checked after the execution of each START statement.

IX211

X-Numbers: Basic set plus 24, 44, 69, 74, and 75.

Features Tested: IX211 tests the capability to change (update) index keys of records in an Indexed file and then retrieve the records from the file in the proper sequence. A fundamental assumption in testing this capability is that between the OPENing of a file for processing and the CLOSE of that file there is no reordering of the indexed keys for the file. The records are retrieved sequentially from the file as if no updating had taken place. When the file is reopened for processing the records that were updated are then expected to be sequentially accessed in the updated order.

Record modification for the file involves the updating of the update-number field, the record-key or alternate-key, and the ODO-Number field of the record. Each time a given record is modified the update-number field will be incremented by one. To keep track of those record modified, the ODO-number field will always carry the sequential location of the record within the file which reflects the last key value position before the record was modified. This location number will be used for verifying that the sequential reordering of the record for the file was accomplished successfully. Only one of the 3 record keys of the record will be modified for any given REWRITE of the record. A test is also made to insure that the current record pointer is not affected by execution of the REWRITE statment.

IX441

X-Numbers: Basic set plus 14, 24, 25, 26, 53, 69, 74, 76, 77, 78.

Features Tested: The program IX441 contains the level 1 Indexed I-O language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The Indexed I-O level 1 elements in this program are contained in the High level of Federal Standard COBOL. All Indexed I-O level 1 language features are tested in this program.

IX442

X-Numbers: Basic set plus 24, 25, 69, 74, 75, 86.

Features Tested: The program IX442 contains the level 2 Indexed I-O language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The Indexed I-O level 2 elements in this program are contained in the High level of Federal Standard COBOL. All Indexed I-O level 2 language features are tested in this program. In addition, if this program is compiled specifying the Low, Low-Intermediate, or High-Intermediate levels of Federal Standard COBOL, then all Indexed I-O level 1 elements in the program will also be flagged.

D.5 LIBRARY MODULE

LB101

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the use of the COPY statement in a File Description and its related 01 entries, in the WORKING-STORAGE SECTION, and in the PROCEDURE DIVISION. This program creates a sequential file of 7 records that is used as input by the program LB102 which checks the file to assure proper execution of the COPY function in LB101. The library entries copied into LB101 include:

K101A
K1F0A
K1W01
K1WC2
K1W03
K1W04
KW1KA

LB102

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the output file of LB101 to assure proper execution of the COPY statements in LB101.

LB103

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates a sequential file which is read and checked in LB104. The two programs in combination test the COPY function for the ENVIRONMENT DIVISION, SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, FILE-CONTROL and I-O-CONTROL entries. The SPECIAL-NAMES feature is tested thru the use of DECIMAL-POINT IS COMMA, and the I-O-CONTROL paragraph is tested thru the use of the SAME AREA clause. The library entries copied into LB103 include:

K3FCA
K3IOA
K3OCA
K3SCA
K3SNA

LB104

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the output of program LB103 for proper execution of COPY of SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, FILE-CONTROL and I-O-CONTROL. Proper execution of the COPY statements is assumed if the sequential file created in program LB103 agrees with the expected results.

LB105

X-Numbers: Basic set plus 01, 27, 69, 74, 75.

Features Tested: This program tests the use of the COPY statement in the DATA DIVISION for a Sort Description entry and the related record description entries. The library entries copied into LB105 include:

K501A
K5SDA

LB106

X-Numbers: Basic set only.

Features Tested: This program contains a COPY statement in the line following the ENVIRONMENT DIVISION header. The remainder of the ENVIRONMENT DIVISION and the entire DATA and PROCEDURE DIVISIONs are copied by the single COPY statement in this program. The library entries copied into LB106 include:

K6SCA

LB107

X-Numbers: Basic set only.

Features Tested: This program tests the capability to perform a COPY of 1599 source lines into the PROCEDURE DIVISION. The library entries copied into LB107 include:

K7SEA

LB201

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the use of the COPY REPLACING statement in the WORKING-STORAGE SECTION and the PROCEDURE DIVISION. LB201 also tests the COPY REPLACING statement for the File Descrip-

tion and its related 01 entries. A 7 record sequential file is created in LB201 and passed to LB202 for processing.

Note that WORKING-STORAGE records WSTR-3 and WSTR-5 copy library data which is referenced elsewhere. A compiler may require that these COPY statements be changed to COPY K1WKY and COPY K1WK2 respectively. The necessity of making such changes constitutes test failures. The library entries copied into LB201 include:

K101A
K1FDA
K1PRB
K1WKA
K1WKB

LB202

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the output file of proram LB201 to assure proper execution of the COPY REPLACING of the FD and related 01 entries for a sequential file. The program also checks the COPY REPLACING statement where the words being replaced are paragraph-names, one in an ALTER statement, one in a PERFORM, and one a numeric paragraph-name. Also checked is the ability to COPY with replacement by literal, both alphanumeric and numeric. The library entries copied into LB202 include:

K2PRA
K7SEA

LB203

X-Numbers: Basic set plus 02, 69, 74, 77.

Features Tested: This program creates a sequential file which is checked in program LB204. The two programs test the COPY REPLACING statement for the ENVIRONMENT DIVISION, FILE-CONTROL and I-O-CONTROL entries. The library entries copied into LB203 include:

K3FCB

LB204

X-Numbers: Basic set plus 02, 69, 74, 77.

Features Tested: This program checks the output of program LB203 for proper execution of the COPY REPLACING of SOURCE-COMPUTER, OBJECT-COMPUTER, FILE-CONTROL, and I-O-CONTROL. Proper execution is assumed if the sequential file created in LB203 agrees with

the expected results. Program LB204 is similar to LB104 except that SPECIAL-NAMES is not tested.

LB205

X-Numbers: Basic set plus 01, 27, 69, 74, 76.

Features Tested: LB205 tests the COPY REPLACING statement for a Sort Description and its related 01 entries. The library entries copied into LB205 include:

K501B
K5SDB

LB206

X-Numbers: Basic set only.

Features Tested: This program exercises pseudo-text replacement in the PROCEDURE DIVISION. The library entries copied into LB206 include:

KP001
KP002
KP003
KP004
KP005
KP006
KP007
KP008

LB207

X-Numbers: Basic set plus 47.

Features Tested: This program tests the COPY statement using a qualified text-name. Two tests are executed each involving a COPY statement referencing the same text-name, but each from a different library-name.

When the libraries are created in preparation for running LB207 the text which will be placed in the library equated to X-47 should be selected from the population file using the Plus-Card '+ALTLB'. The text which will be placed in the library equated to X-48 should be selected from the population file using the Plus-Card '+ALTL1,,,ALTLB'.

LB421

X-Numbers: Basic set only.

Features Tested: The program LB421 contains level 1 Library language features

for purposes of testing the FIPS PUB 21-1 flagging requirement. The LIB level 1 elements in this program are contained in Low-Intermediate level of Federal Standard COBOL. The Procedure Divison statement to be flagged is:

COPY text-name

LB441

X-Numbers: Basic set plus 47.

Features Tested: The Program LB441 contains level 2 Library language features for purposes of testing the FIPS PIB 21-1 flagging requirement. The LIB level 2 elements in this program are contained in High level of Federal Standard COBOL. The Procedure Division statement to be flagged is:

COPY text-name of library-name REPLACING...BY...

D.6 NUCLEUS MODULE

NC101

X-Numbers: Basic set only.

Features Tested: This program tests the MULTIPLY and DIVIDE statements.

NC102

X-Numbers: Basic set only.

Features Tested: This program tests the use of comment lines and the GO TO, ALTER, EXIT and PERFORM statements.

NC103

X-Numbers: Basic set plus 51, 52.

Features Tested: This program tests the following features:

IF
Level Numbers
Switch Status Conditions
Relation Conditions
Class Conditions
Initialization of Items (exercised in IF tests)

NC104

X-Numbers: Basic set only.

Features Tested: This program test the MOVE statement.

NC105

X-Numbers: Basic set only.

Features Tested: This program continues the tests of the MOVE statement which were begun in NC104.

NC106

X-Numbers: Basic set only.

Features Tested: This program tests the ADD and SUBTRACT statements with the optional ROUNDED and ON SIZE ERROR phrases. Truncation of resultant data items and the EXAMINE statement are also tested.

NC107

X-Numbers: Basic set only.

Features Tested: This program tests the following COBOL features:

Figurative Constants
Continuation
JUSTIFIED
SYNCHRONIZED
BLANK WHEN ZERO
Long-Names and Literals
REDEFINES
USAGE
SPECIAL-NAMES
CURRENCY SIGN IS
DECIMAL-POINT IS COMMA
Reserve Words in Remarks

NC108

X-Numbers: Basic set plus 51.

Features Tested: This program tests the following COBOL features:

Abbreviations
Compact IDENTIFICATION DIVISION
Switches
CURRENCY IS
PIC
COMP
JUST
SYNC
BLANK ZERO
COBOL Character Set
Complete Data Format

NC109

X-Numbers: Basic set only.

Features Tested: This program tests the STOP literal, ACCEPT, and DISPLAY statements.

Special Considerations: The tests for the ACCEPT statement process data from the system input device in the following order:

Entry 1:ABCDEFGHIJKLMNOPQRSTUVWXY Z
Entry 2:0123456789
Entry 3:().+-*/$, =
Entry 4:9
Entry 5:0
Entry 6: ABC XYZ
Entry 7:0123456789
Entry 8:single space

Entry 9:*
Entry 10:ABCD
Entry 11:A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0123456789

NC110

X-Number: Basic set only.

Features Tested: The PROCEDURE DIVISION consists entirely of paragraph-names and DISPLAY literal statements. The entire report should print on the system output device. If a copy of the output report cannot be printed on the output device, a visual inspection must be made of the display device to ensure the validity of the tests.

NC111

X-Numbers: Basic set only.

Features Tested: This program tests the ADD and SUBTRACT statements with the optional ROUNDED and ON SIZE ERROR phrases. Truncation of resultant data items and the EXAMINE statement are also tested.

NC112

X-Numbers: Basic set only.

Features Tested: This program tests the use of multiple operands in ADD, SUBTRACT, and MOVE statements.

NC113

X-Numbers: Basic set only.

Features Tested: This program verifies that a statement which must begin in Area A can start in any of the character positions 8 through 11.

NC114

X-Numbers: Basic set only.

Features Tested: This program tests the use of stroke, alphabetic and numeric editing. The GO TO, IF, MOVE, and PERFORM statements are also tested.

NC115

X-Numbers: Basic set only.

Features Tested: This program tests the INSPECT statement.

NC116

X-Numbers: Basic set only.

Features Tested: This program tests the SIGN clause and its optional clauses.

NC117

X-Numbers: Basic set only.

Features Tested: This program tests the MULTIPLY and DIVIDE statements which contain operands defined with the SIGN clause.

NC118

X-Numbers: Basic set only.

Features Tested: This program tests the ADD statement with the optional clauses ROUNDED and ON SIZE ERROR. Truncation and the EXAMINE statement are also tested. The operands in these tests are defined with the SIGN clause.

NC119

X-Numbers: Basic set only.

Features Tested: This program tests the SUBTRACT statement with the optional ROUNDED and ON SIZE ERROR phrases. The operands in these tests are defined with the SIGN clause and truncation of the result is required.

NC120

X-Numbers: Basic set only.

Features Tested: This program tests the MULTIPLY statement which contains operands defined with the SIGN clause.

NC151

X-Numbers: Basic set only.

Features Tested: This program tests the MULTIPLY and DIVIDE statements.

NC152

X-Numbers: Basic set only.

Features Tested: This program tests the GO, EXIT and PERFORM statements.

NC153

X-Numbers: Basic set only.

Features Tested: This program tests the IF statement.

NC154

X-Numbers: Basic set only.

Features Tested: This program continues the IF statement tests begun in NC153.

NC155

X-Numbers: Basic set only.

Features Tested: This program tests the following features:

Level Numbers
Relation Conditions
Class Conditions
Initialization of Items

NC156

X-Numbers: Basic set only.

Features Tested: This program tests the ADD and SUBTRACT statements with the optional ROUNDED and SIZE ERROR phrases. Truncation of the resultant data items are also tested.

NC157

X-Numbers: Basic set only.

Features Tested: This program tests the following COBOL features:

Continuation
SYNC
Long Names and Literals
REDEFINES
USAGE
Reserve Words in Remarks

NC158

X-Numbers: Basic set only.

Features Tested: This program tests the STOP literal, ACCEPT, and DISPLAY statements.

Special Considerations: The tests for the ACCEPT statement process data from the system input device in the following order:

```
Entry 1:ABCDEFGHIJKLMNOPQRSTUVWXY Z
Entry 2:0123456789
Entry 3:().+-*/$,
Entry 4:9
Entry 5:0
Entry 6: ABC              XYZ
Entry 7:012345678
Entry 8:single space
Entry 9:'
Entry 10:ABCD
```

NC159

X-Numbers: Basic set only.

Features Tested: The PROCEDURE DIVISION consists entirely of paragraph-names and DISPLAY statements. The entire report should print on the system output device. If a copy of the output report cannot be printed and saved on the output device a visual inspection must be made of the display device to ensure the validity of the tests.

NC160

X-Numbers: Basic set only.

Features Tested: This program tests the use of stroke, alphabetic and numeric editing. The GO, IF, MOVE, and PERFORM statements are also tested.

NC161

X-Numbers: Basic set only.

Features Tested: This program tests the MOVE statement.

NC162

X-Numbers: Basic set only.

Features Tested: This program continues the tests of the MOVE statement which were begun in NC161.

NC163

X-Numbers: Basic set only.

Features Tested: This program continues the MOVE statement tests of NC161 and NC162.

NC164

X-Numbers: Basic set only.

Features Tested: This program continues the MOVE statement tests of NC161, NC162, and NC163.

NC165

X-Numbers: Basic set only.

Features Tested: This program continues the MOVE statement tests of NC161, NC162, NC163, and NC164.

NC201

X-Numbers: Basic set only.

Features Tested: This program tests the PERFORM and IF statements with qualified data-names and condition-names. The GO TO statement without a procedure-name is tested in conjunction with the ALTER statement.

NC202

X-Numbers: Basic set only.

Features Tested: This program tests the ADD CORRESPONDING and SUBTRACT CORRESPONDING statements. The ADD and SUBTRACT statements with multiple resultant data items in the GIVING phrase are also tested.

NC203

X-Numbers: Basic set only.

Features Tested: This program tests the DATA DIVISION clauses REDEFINES and RENAMES as well as level numbers. The DIVIDE statement

is also tested.

NC204

X-Numbers: Basic set plus 56, 57.

Features Tested: This program tests the DISPLAY UPON and ACCEPT UPON statements.

Special Considerations: The tests for the ACCEPT statements process data from the system input device in the following order:

```
Entry 1:ABCDEFGHIJKLMNOPQRSTUVWXY Z
Entry 2:0123456789
Entry 3:().+-*/$, =
Entry 4:9
Entry 5:0
Entry 6: ABC              XYZ
Entry 7:single space
Entry 8:'
Entry 9:Q
Entry 10:ABCD
Entry 11:ABCD
Entry 12:A B C D E F G H I J K L M N O P
           Q R S T U V W X Y Z  0123456789
Entry 13:0001*002*003*004*005*006*007*008
           *009*010*011*012*013*014*015*016
           *017*018*019*020*021*022*023*024
           *025*026*027*028*029*030*031*032
           *033*034*035*036*037*038*039*040
           *041*042*043*044*045*046*047*048
           *049*050
```

NC205

X-Numbers: Basic set only.

Features Tested: This program tests continuation of lines.

NC206

X-Numbers: Basic set only.

Features Tested: This program tests qualification including qualified subscripts.

NC207

X-Numbers: Basic set only.

Features Tested: This program tests ADD, SUBTRACT, MULTIPLY, and DIVIDE

statements which contain qualified data-names as operands.

NC208

X-Numbers: Basic set only.

Features Tested: This program contains qualified paragraph-names and MOVE statements with qualified data-names as operands. The MOVE statement with multiple receiving areas is also tested.

NC209

X-Numbers: Basic set only.

Features Tested: This program continues the tests of the MOVE CORRESPONDING statement and the MOVE statement with multiple receiving areas which were begun in NC208.

NC210

X-Numbers: Basic set only.

Features Tested: This program tests nested IF statements.

NC211

X-Numbers: Basic set only.

Features Tested: This program tests the IF statement with qualified data-names, condition-names, and with compound conditional statements. The PERFORM statement and the GO TO statement without a procedure-name are tested in conjunction with the ALTER statement.

NC212

X-Numbers: Basic set only.

Features Tested: This program tests 49 levels of group and elementary defined items.

NC213

X-Numbers: Basic set only.

Features Tested: This program tests format 5 of the DIVIDE statement and the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements with multiple resultant data items.

NC214

X-Numbers: Basic set only.

Features Tested: This program tests the use of a double quote in a literal, logical connectives, and the ACCEPT FROM statement for DATE, DAY, and TIME. The PERFORM VARYING statement is also tested in this program.

NC215

X-Numbers: Basic set only.

Features Tested: This program tests the use of a user-defined collating sequence in the alphabet-name clause of the SPECIAL-NAMES paragraph. The PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph references the alphabet-name defined in SPECIAL-NAMES.

NC216

X-Numbers: Basic set only.

Features Tested: This program tests the level 2 INSPECT statement features.

NC217

X-Numbers: Basic set only.

Features Tested: This program tests the STRING statement.

NC218

X-Numbers: Basic set only.

Features Tested: This program tests the UNSTRING statement.

NC219

X-Numbers: Basic set only.

Features Tested: This program tests the PROGRAM COLLATING SEQUENCE clause and the ALPHABET-NAME clause. The program is designed to test the effect of using the HIGH-VALUE and LOW-VALUE figurative constants within the literals for the ALPHABET-NAME clause associated with the PROGRAM COLLATING SEQUENCE statement.

The actual statements used are shown below:

```
PROGRAM COLLATING SEQUENCE IS COLLATING-SEQ-1.

SPECIAL-NAMES.

     COLLATING-SEQ-1 IS "F" "U" "N"

          ALSO HIGH-VALUE

          ALSO LOW-VALUE

          "Y"

     COLLATING-SEQ-2 IS "Q"

          ALSO HIGH-VALUE

          ALSO LOW-VALUE.
```

The program consists of tests which compare the relative ordinal positions of the characters as described in the ALPHABET-NAME clauses.

NC431

X-Numbers: Basic set only.

Features Tested: The program NC431 contains the level 2 Nucleus language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The Nucleus level 2 elements is this program are contained in the High-Intermediate and High levels of Federal Standard COBOL.

One Identification Division statement should be flagged.

One Environment Division statement should be flagged.

Twelve (12) "areas" should be flagged in the Data Division.

Forty-nine (49) "areas" should be flagged in the Procedure Division.

D.7 RELATIVE I-O MODULE

RL101

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length relative file whose access mode is sequential. The record size is 120 characters. Upon completion of this program the relative file is passed to RL102 for further processing.

RL102

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program is used to verify and update the 500 record fixed length relative file created in RL101. The access mode for this file in this program is random. After the file is initially verified for accuracy of data it is opened as I-O and every fifth record is updated. Following this operation the file is again verified for correctness. Upon completion of the program the file is passed to RL103 for further processing.

RL103

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program is used to verify the 500 record fixed length file updated in RL102. The access mode for this file in this program is sequential. After the file is initially verified for accuracy of data it is opened as I-O and every fourth record is deleted. Following this the file is again verified for correctness. There should be 375 records remaining in the file.

RL104

X-Numbers: Basic set plus 22, 69,74, 75.

Features Tested: This program creates and updates a 500 record fixed length relative file whose access mode is sequential. The record size is 120 characters. After the file is created it is opened as I-O and every fifth record is updated. All READ, WRITE, and REWRITE statements are used without the appropriate AT END or INVALID KEY phrases. A USE procedure for file-name is specified and the contents of the FILE STATUS data item is tested following each I-O operation.

RL105

X-Numbers: Basic set plus 21, 22, 23, 69, 74, 75, 76, 77.

Features Tested: This program creates and verifies three fixed length records relative files whose access mode is sequential. The three files contain 19, 16, and 16 records respectively. The record size for all files is 100 characters.

RL106

X-Numbers: Basic set plus 21, 22, 23, 69, 74, 75, 76, 77.

Featues Tested: This program creates and verifies three variable length record relative files whose access mode is random. The three files contain 18, 10, and 12 records respectively. Record lengths are 56, 100, or 102 characters. The RELATIVE KEY data item is used to access specific records within each of the three files and to verify their accuracy.

RL107

X-Numbers: Basic set plus 21, 22, 69, 74, 75, 76.

Features Tested: This program creates and verifies two fixed length relative files whose access mode is random. One file contains 125 records and the other file contains 25 records. The record size for both files is 120 characters. The first (or larger) file is only partially created during the OUTPUT mode but is subsequently completed in the I-O mode. The file is not created in sequential order but the end result is that there are no null records in the file. Various records in both files are tested for accuracy while using the RELATIVE KEY data items to access the specific records to be tested. The first file is tested while opened for I-O and the second file is tested while opened for INPUT. A test is also made to read a record which has yet to be created.

RL108

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length relative file whose access mode is sequential. The record length is 240 characters. Upon completion of the program the relative file is passed to RL109 for further processing.

RL109

X-Numbers: Basic set plus 01, 21, 51, 53, 69, 74, 75, 76.

Features Tested: This program is designed to test the use of the RERUN clause with a program which processes a relative file. This program reads and verifies the 500 record file created in RL108. The access mode for this file is sequential. At least one of the following forms of the RERUN clause must be implemented:

RERUN ON implementor-name EVERY integer RECORDS OF file-name-2
RERUN ON implementor-name EVERY integer CLOCK-UNITS

RERUN ON implementor-name EVERY condition-name
RERUN ON file-name-1 EVERY condition-name

The condition to cause the rerun information to be recorded must occur at least once during the execution of this program.

RL151

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length relative file whose access mode is sequential. The record length is 120 characters. The OPEN, CLOSE, READ, and WRITE statements are tested according to the HYPO-COBOL specifications. Upon completion of this program the relative file is passed to RL151 for further processing.

RL152

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program is used to verify and update the 500 record fixed length relative file created in RL151. The access mode for this file in this program is random. After the file is initially verified for accuracy of data it is opened as I-O and every fifth record is updated. Following this operation the file is again verified for correctness. The OPEN, CLOSE, READ, and REWRITE statements are tested according to the HYPO-COBOL specifications. Upon completion of this program the relative file is passed to RL153 for further processing.

RL153

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program is used to verify the 500 record fixed length file updated in RL152. The access mode for this file in this program is sequential. After the file is initially verified for accuracy of data it is opened as I-O and every fourth record is deleted. Following this operation the file is again verified for correctness. There should be 375 records remaining in the file. The OPEN, CLOSE, READ, and DELETE statements are tested according to the HYPO-COBOL specifications.

RL201

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program creates and verifies a 500 record fixed length relative file whose access mode is sequential. The record size is 120 characters. Upon completion of this program the relative file is passed to RL202 for further processing.

RL202

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program is used to verify and update the 500 record fixed length relative file created in RL201. The access mode for this file in this program is dynamic. After the file is initially verified for accuracy of data it is opened as I-O and every fifth record is updated. Following this operation the file is again verified for correctness. The RELATIVE KEY data item is used to access the next record to be read or updated. The verification of the initial input file is performed in ascending sequence and the verification of the updated file is performed in descending sequence. Upon completion of this program the updated relative file is passed to RL203 for further processing.

RL203

X-Numbers: Basic set plus 21, 69, 74, 75.

Features Tested: This program is used to verify the 500 record fixed length relative file updated in RL202. The access mode for this file in this program is dynamic. After the file is initially verified for accuracy of data it is opened as I-O and every fourth record is deleted. Following this operation the file is again verified for correctness. There should be 375 records remaining in the file. All read operations are performed using the READ NEXT RECORD statement.

RL204

X-Numbers: Basic set plus 22, 69, 74, 76.

Features Tested: This program creates and updates a 500 record fixed length relative file whose access mode is dynamic. The record size is 120 characters. After the file is created it is opened as I-O and every fifth record is updated. All READ, WRITE, and REWRITE statements are used without the appropriate AT END or INVALID KEY phrases. A USE procedure for file-name is specified and the contents of the FILE STATUS data item is tested following each I-O operation.

RL205

X-Numbers: Basic set plus 21, 22, 69, 74, 75, 75.

Features Tested: This program creates and verifies two 300 record fixed length relative files. One file is created with access mode is dynamic and the other with access mode is sequential. The record length for both files is 240 characters. This program is designed to test the use of the READ and START statements. Various syntactical constructs of the READ statement are used to sequentially and randomly verify specific records on the dynamic accessed file. Also both files are processed

sequentially using various syntactical constructs of the START statement to logically position within the file.

RL421

X-Numbers: Basic set plus 14, 21, 22, 23, 53, 69, 74, 75, 76, 77, 78.

Features Tested: The program RL421 contains the level 1 Relative I-O language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The Relative I-O level 1 elements in this program are contained in the Low-Intermediate level of Federal Standard COBOL. All Relative I-O level 1 language features are tested in this program.

RL431

X-Numbers: Basic set plus 21, 22, 69, 74, 75, 86.

Features Tested: The program RL 431 contains the level 2 Relative I-O language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The Relative I-O level 2 elements in this program were contained in the High-Intermediate and High levels of Federal Standard COBOL. All Relative I-O level 2 language features are tested in this program. In addition, if this program is compiled specifying the Low level of Federal Standard COBOL, then all Relative I-O level 1 elements in this program will be flagged.

D.8 REPORT WRITER MODULE

RW101

X-Numbers: Basic set plus 49.

Features Tested: The routine RW101 tests basic Report Writer module functions. The report description in this routine contains a PAGE LIMIT IS 20 LINES without the optional HEADING, FIRST DETAIL, LAST DETAIL or FOOTING phrases. A single Detail Report Group is defined for the report. An output report in the usual audit routine format is produced using WRITE statements, and a one page report of 20 lines is produced by the RWCS. The sequential file RW-FS1 is the report file and is assigned to the RWCS output device through X-number 49.

RW102

X-numbers: Basic set plus 49.

Features Tested: The routine RW102 tests basic Report Writer module functions. The report description in this routine contains PAGE LIMIT 20, FIRST DETAIL 1, LAST DETAIL 25 without the optional HEADING or FOOTING phrases. A single Detail Report Group is defined for the report. An output report in the usual audit routine format is produced using WRITE statements, and a one page report of 20 lines is produced by the RWCS. The sequential file RW-FS2 is the report file and is assigned to the RWCS output device through X-number 49.

RW103

X-Numbers: Basic set plus 49.

Features Tested: The routine RW103 tests basic Report Writer module functions. The report description in this routine contains PAGE 30, HEADING 1, FIRST DETAIL 6, LAST DETAIL 25 without the optional FOOTING phrase. A Page Heading Report Group and a Detail Report Group are defined for the report. An output report in the usual audit routine format is produced using WRITE statements, and a three page report of 1 page heading line and twenty detail lines on each page is produced by the RWCS. The sequential file RW-FS3 is the report file and is assigned to the RWCS output device through X-number 49.

RW104

X-Numbers: Basic set plus 49.

Features Tested: The routine RW104 tests basic Report Writer module functions.

The report description in this routine contains PAGE LIMITS ARE 30 LINES, HEADING 1, FIRST DETAIL 6, LAST DETAIL 25, FOOTING 29. A Page Heading, Page Footing and one Detail Report Group are defined for the report. An output report in the usual audit routine format is produced using WRITE statements, and a three page report with 1 page heading line, 1 page footing line and twenty detail lines on each page is produced by the RWCS. The sequential file RW-FS4 is the report file and is assigned to the RWCS output device through X-number 49.

D.9 SEGMENTATION MODULE

SG101

X-Numbers: Basic set only.

Features Tested: This program exercises transfer of control via the PERFORM statement to various fixed permanent segments and independent segments from fixed permanent segments. Control is also passed from independent segments to fixed permanent segments. There are 50 fixed permanent segments and 50 independent segments. There is no attempt to test either the last used or the initial state in this program.

SG102

X-Numbers: Basic set only.

Features Tested: This program exercises various ALTER and PERFORM statements and prepares a directory in each test to trace program flow. Transfers of control are from fixed permanent segments to fixed permanent segments, fixed permanent segments to independent segments, independent segments to fixed permanent segments, and independent segments to independent segments. There are no tests for the initial state per se, but there are tests to ensure that the last used state is present when required.

SG103

X-Numbers: Basic set only.

Features Tested: This program uses the ALTER, PERFORM and GO TO statements to check initial and last used segment states. Transfers of control are from fixed permanent segments to fixed permanent segments, fixed permanent segments to independent segments, independent segments to fixed permanent segments, and independent segments to independent segments. The form of passing control is through the use of PERFORM and GO TO statements and fall-through logic.

SG104

X-Numbers: Basic set plus 27.

Features Tested: This program tests the compiler's ability to segment a program using a SORT statement which requires both an INPUT PROCEDURE and an OUTPUT PROCEDURE. The sections containing the SORT statement, the INPUT PROCEDURE and the OUTPUT PROCEDURE are all in the same independent segment. The results for this

program are identical to the unsegmented ST108.

SG105

X-Numbers: Basic set plus 27.

Features Tested: This program tests the compiler's ability to segment a program SORT statement which requires both an INPUT PROCEDURE and an OUTPUT PROCEDURE. The section containing the SORT statement is in an independent segment while the INPUT and OUTPUT PROCEDUREs are located in fixed permanent segments. The results are identical to those expected for the unsegmented ST108.

SG106

X-Numbers: Basic set plus 27.

Features Tested: This program tests the compiler's ability to segment using a SORT statement which requires both an INPUT PROCEDURE and an OUTPUT PROCEDURE. The section containing the SORT statement is in a fixed permanent segment while the INPUT and OUTPUT PROCEDUREs are located in different independent segments. The results are identical to those expected for ST108.

SG201

X-Numbers: Basic set only.

Features Tested: This program tests the initial state of independent segments and the last used state of fixed overlayable segments. A SEGMENT-LIMIT of 30 is specified in the OBJECT-COMPUTER paragraph thereby causing segments from 30 to 49 to become fixed overlayable segments. The first section encountered in the PROCEDURE DIVISION is an independent segment. There are 100 segments present in this program.

SG202

X-Numbers: Basic set only.

Features Tested: This program contains tests designed to alter fixed overlayable segments that have not yet been called for execution, to test code which falls through to independent segments, and to PERFORM fixed overlayable segments. Due to a SEGMENT-LIMIT of 25, segments 25 through 49 are considered fixed overlayable segments. This program contains no independent segments.

SG203

X-Numbers: Basic set only.

Features Tested: This program tests the initial state for independent segments and the last used state for fixed overlayable segments. The PERFORM and GO TO statements and the code which falls through to an independent segment are used to determine whether the initial state or last used state is found in each segment.

SG204

X-Numbers: Basic set plus 01, 14, 15, 27, 28, 29, 69, 74, 75, 76, 77.

Features Tested: This program tests the compiler's ability to segment a program using three SORT statements consisting of a variety of INPUT and OUTPUT PROCEDUREs as well as the USING and GIVING phrases. The SORT statements and all related INPUT and OUTPUT PROCEDUREs are located in fixed permanent segments, fixed overlayable segments, and independent segments. The results are identical to those expected for the unsegmented ST201.

SG421

X-Numbers: Basic set only.

Features Tested: The program SG421 contains the level 1 Segmentation language features for purposes of testing the FIPS PIB 21-1 flagging requirements. The Segmentation level 1 elements in this program are contained in Low-Intermediate and High-Intermediate levels of the Federal Standard COBOL. The Procedure Division statement to be flagged is:

segment-name SECTION segment-number

SG441

X-Number: Basic set only.

Features Tested: The program SG441 contains the level 2 Segmentation language features for purposes of testing the FIPS PUB 21-1 flagging requirements. The Segmentation level 2 elements in this program are contained in the High level of the Federal Standard COBOL. The Environment Division element to be flagged is:

SEGMENT-LIMIT IS segment-number

D.10 SEQUENTIAL I-O MODULE

SQ101

X-Numbers: Basic set only.

Features Tested: All combinations of the level 1 WRITE statement for a printer output file are tested in this program. Tests of the line overprint capability and tests which assure that column 1 does not act as a carriage control field are also included.

SQ102

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates and verifies a magnetic tape file containing 750 fixed length records of 120 characters. The OPEN, CLOSE, READ, and WRITE statements are tested for level 1 features.

SQ103

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates and verifies magnetic tape file containing 500 fixed length records of 120 characters. The File-Control entry for the file contains a File Status clause and a USE statement defines the procedures to be executed when an end-of-file is encountered.

SQ104

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies mass storage file containing 649 fixed length records of 120 characters. The OPEN, CLOSE, READ, and WRITE statements are tested for level 1 features.

SQ105

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies a mass storage file containing 980 fixed length records of 125 characters. There are two USE procedures in the DECLARATIVE SECTION, one for exceptions on output, the other for exceptions on input. The end-of-file condition should cause execution of the USE procedure for exception on input since the AT

END phrase is not included in any READ statement.

SQ106

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates and verifies a magnetic tape file containing 450 variable length records of 120 or 151 characters. The OPEN, CLOSE, READ, and WRITE statements are tested for level 1 features.

SQ107

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies a mass storage file containing 450 variable length records of 120 or 151 characters. The OPEN, CLOSE, READ, and WRITE statements are tested for level 1 features.

SQ108

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies a mass storage file containing 710 fixed length records of 141 characters. The various forms of the READ INTO statement are tested in this program.

SQ109

X-Numbers: Basic set plus 06, 69, 74, 75.

Features Tested: This program creates and verifies multi-reel magentic tape file containing 750 fixed length records of 120 characters. The CLOSE REEL option is used to create a two reel file. There are 325 records on the first reel and 425 records on the second reel.

Special Considerations:

(1) A two reel magnetic tape file is required if allowed by the operating system.
(2) The H-option is used to include all CLOSE REEL statements in the source program when a multi-reel file is specified.
(3) The I-option is used to delete all CLOSE REEL statements from the source program when a multi-reel file is not specified.

SQ110

X-Numbers: Basic set plus 19, 69, 74, 75.

Features Tested: This program creates and verifies a multi-unit mass storage file containing 649 fixed length records of 120 characters. The CLOSE UNIT option is used to create a two unit file. There are 196 records on the first unit and 453 records on the second unit.

Special Considerations:

(1) A two unit mass storage file is required if allowed by the operating system.
(2) The E-option is used to include all CLOSE UNIT statements in the source program when a multi-unit file is specified.
(3) The F-option is used to delete all CLOSE UNIT statements from the source program when a multi-unit file is not specified.

SQ111

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates and verifies a magnetic tape file which includes a CODE-SET clause in the File Description entry. The file contains 595 fixed length records of 155 characters per record. The following COBOL features are included in this program:

```
SPECIAL-NAMES.
    alphabet-name IS STANDARD-1.
CODE-SET IS alphabet-name
Record desciption for the file contains
    SIGN IS LEADING SEPARATE CHARACTER
```

SQ112

X-Numbers: Basic set plus 19, 69, 74, 75.

Features Tested: This program creates and verifies a multi-unit mass storage file containing 500 fixed length records of 120 characters. The first unit contains 200 records and the second unit contains 300 records. The two unit file is passed to SQ113 for further processing.

Special Considerations:

(1) A two unit mass storage file is required if allowed by the operating system.
(2) The E-option is used to include all CLOSE UNIT statements in the source program when a multi-unit file is specified.
(3) The F-option is used to delete all CLOSE UNIT statements

from the source program when a multi-unit file is not specified.
(4) The output file generated by SQ112 is passed to SQ113 for further processing.

SQ113

X-Numbers: Basic set plus 01, 19, 51, 53, 69, 74, 75, 76.

Features Tested: The program SQ113 tests the RERUN clause. The two unit mass storage file created in SQ112 is verified by this program. The following COBOL features are included in the program:

```
SPECIAL-NAMES.
    implementor-name IS mnemonic-name
        ON STATUS IS condition-name-1
        OFF STATUS IS condition-name-2.
I-O-CONTROL.
    RERUN ... (use any of the 7 optional formats on X-card 53)
```

SQ114

X-Numbers: Basic set plus 01, 14, 69, 74, 75.

Features Tested: This program creates and verifies a fixed length magnetic tape file and a fixed length mass storage file. The magnetic tape file contains 750 fixed length records of 120 characters. The mass storage file contained 649 fixed length records of 120 characters. The SAME AREA clause is specified for the two files.

SQ115

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies a mass storage file containing 550 fixed length records of 126 characters. The file is then opened as an input-output file and every tenth record is rewritten. The updated file is verified to ensure the REWRITE statements executed correctly.

SQ116

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies a mass storage file containing 550 fixed length records of 130 characters. The file is then opened as an input-output file and records are updated using REWRITE FROM statements. The updated file is verified to ensure the REWRITE FROM statements executed correctly.

SQ117

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This routine creates a mass storage file using WRITE FROM statements. The file contains 495 fixed length records of 141 characters. The file is opened as in the input mode and records are checked to ensure the WRITE FROM statements executed correctly.

SQ118

X-Numbers: Basic set plus 01, 54, 69, 74, 75, 76.

Features Tested: This program is designed to test the transportability of ASCII data and signed numeric data between computer systems. A magnetic tape file containing fifty 120 character records and a punched card file containing fifty 80-column cards are created. Each tape and card record contains a signed numeric data item with leading sign in character positions 12 through 27 and a signed numeric data item with trailing sign in character positions 41 through 57. The ASCII character set is generated in collating sequence order, one character type per record, in character positions 115 through 120 for tape and columns 75 through 80 for cards. The program includes the following COBOL features:

```
SPECIAL-NAMES.
    alphabet-name IS STANDARD-1.
CODE-SET IS alphabet-name
CODE-SET alphabet-name
SIGN IS LEADING SEPARATE CHARACTER (in record description)
SIGN IS TRAILING SEPARATE CHARACTER (in record description)
COBOL character set (except for quote)
```

Special Considerations:

Upon completion of the validation the output tape and card files are to be returned to FCCTS for processing and verfication on a foreign computer system.

SQ119

X-Numbers: Basic set plus 01, 58, 69, 74, 75, 76.

Features Tested: This program is designed to test the transportability of ASCII data and signed numeric data between computer systems. SQ119 reads and validates a magnetic tape file and a punched card file created by FCCTS on a foreign computer system. The tape file contains fifty 120-character records and the card file contains fifty 80-column cards. Each tape and card

record contains a signed numeric data item with leading sign in character positions 12 through 27 and a signed numeric data item in character positions 41 through 57. The ASCII character set is contained in collating sequence order, one character type per record, in character positions 115 thru 120 for tape and columns 75 thru 80 for cards. The collating sequence specified in this program is NATIVE. Upon completion of this program all files are passed to SQ120 for further processing. The program includes the following COBOL features:

```
PROGRAM COLLATING SEQUENCE IS NATIVE.
SPECIAL-NAMES.
    alphabet-name IS STANDARD-1
    alphabet-name IS NATIVE
CODE-SET alphabet-name
CODE-SET IS alphabet-name
SIGN IS LEADING SEPARATE CHARACTER (in record description)
SIGN IS TRAILING SEPARATE CHARACTER (in record description)
COBOL character set (except for quote)
```

SQ120

X-Numbers: Basic set plus 01, 58, 69, 74, 75, 76.

Features Tested: This program is designed to test the transportability of ASCII data between computer systems. The signed numeric data tested in SQ119 is ignored during the execution of this run. SQ120 reads and validates a magnetic tape file and a punched card file created by FCCTS on a foreign computer system. The tape file contains fifty 120-character records and the card file contains fifty 80-column cards. Each tape record contains the ASCII character set in collating sequence order, one character type per record, in character positions 115 through 120. For the card file the ASCII data is contained in card columns 75 through 80. The collating sequence specified for this program is ASCII. The program includes the following COBOL features:

```
PROGRAM COLLATING SEQUENCE alphabet-name
SPECIAL-NAMES.
    alphabet-name IS STANDARD-1
CODE-SET alphabet-name
CODE-SET IS alphabet-name
COBOL character set (except for quote)
```

SQ121

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program tests the USE AFTER STANDARD ERROR PROCEDURE

ON I-O statement. A mass storage file containing 550 fixed length records of 126 characters is created and verified. The file is then opened in the input-output mode and every tenth record is rewritten. The updated file is then verified.

SQ151

X-Numbers: Basic set only.

Features Tested: All syntactical combinations of the HYPO-COBOL WRITE statement for a printer output file are tested in this program. Tests of the line overprint capability and tests which assure that column 1 does not act as a carriage control field are also included.

SQ152

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates and verifies a magnetic tape file containing 750 fixed length records of 120 characters. The OPEN, CLOSE, READ, and WRITE statements are tested according to the HYPO-COBOL specifications.

SQ153

X-Numbers: Basic set plus 14, 69, 74, 75.

Features Tested: This program creates and verifies a mass storage file containing 649 fixed length records of 120 characters. The OPEN, CLOSE, READ, and WRITE statements are tested according to the HYPO-COBOL specifications.

SQ201

X-Numbers: Basic set only.

Features Tested: All syntactical combinations of the level 2 WRITE statement using the FROM and ADVANCING phrases with identifier-2 are tested in this program.

SQ202

X-Numbers: Basic set only.

Features Tested: This program tests the level 2 WRITE statement and the File Description for the printer file contains the following LINAGE clause:

LINAGE IS 50 LINES
WITH FOOTING AT 45
LINES AT TOP 10
LINES AT BOTTOM 6

This program also tests:

LINAGE-COUNTER values after the execution of
OPEN OUTPUT...
WRITE...AFTER ADVANCING PAGE
page overflow
WRITE
WRITE...integer LINE
WRITE...identifier-2 LINES
WRITE...end-of-page combinations

SQ203

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program creates and verifies a magnetic tape containing 750 fixed length records of 120 characters. The magnetic tape file generated by SQ203 is passed to SQ204 for further processing.

SQ204

X-Numbers: Basic set plus 01, 03, 17, 18, 69, 74, 75, 76, 77, 78.

Features Tested: This program tests the use of the OPTIONAL clause and the RESERVE integer AREA clause in the SELECT statement. The optional files SQ-FS2 and SQ-FS4 are not present during execution. The following COBOL constructs are included:

SELECT OPTIONAL...
RESERVE...AREAS
RESERVE...AREA
USE AFTER STANDARD EXCEPTION PROCEDURE ON INPUT
READ...; AT END...
READ...

SQ205

X-Numbers: Basic set plus 01, 14, 69, 74, 75.

Features Tested: This program tests the OPEN EXTEND statement for both a magnetic tape file and a mass storage file. A magnetic tape file is created which contains 750 fixed length records of 126 characters. The file is then extended by 250 records and the extended file is verified. The same test is repeated for a mass storage file.

SQ206

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the REVERSED and NO REWIND phrases in the OPEN statement, and the NO REWIND phrase in the CLOSE statement. A magnetic tape file is created which contains 750 fixed length records of 120 characters.

Special Considerations:

(1) The L-option is used to include all OPEN...REVERSED statements in the source program.
(2) The M-option is used to delete all OPEN...REVERSED statements from the source program.

SQ207

X-Numbers: Basic set plus 08, 09, 69, 71, 72, 74, 75, 76, 77, 78, 79, 80.

Features Tested: This program tests the MULTIPLE FILE TAPE clause in the I-O CONTROL paragraph and the NO REWIND phrase in the OPEN and CLOSE statements. Two magnetic tapes, each containing four files, are created and passed to the programs SQ208 and SQ209. Files placed on the magnetic tape specified by X-08 are passed to both SQ208 and SQ209, whereas, files on X-09 are passed to SQ209 only.

SQ208

X-Numbers: Basic set plus 08, 09, 69, 71, 72, 74, 75, 76.

Features Tested: This program verifies four of the eight magnetic tape files created in SQ207 using the MULTIPLE FILE clause. The files are verified in an order which differs from the order in which they were created. The magnetic tape files are passed to SQ209 for further processing.

SQ209

X-Numbers: Basic set plus 08, 09, 69, 74, 75, 77, 78, 79, 80.

Features Tested: This program verifies five of the eight magnetic tape files created in SQ207 using the MULTIPLE FILE clause. The files are verified in an order which differs from the order in which they were created.

SQ210

X-Numbers: Basic set plus 01, 14, 69, 74, 75, 76.

Features Tested: This program tests the USE statement with file-name series, and the FILE STATUS clause in the SELECT clause. A READ statement without the AT END phrase is used to cause the USE procedure to be executed when an end-of-file is encountered. A magnetic tape file is created containing 500 fixed length records of 120 characters, and a mass storage file is created containing 500 fixed length records of 120 characters.

SQ211

X-Numbers: Basic set plus 01, 02, 14, 15, 69, 74, 75, 76, 77, 78.

Features Tested: This program tests the SAME AREA and SAME RECORD AREA clauses in the I-O-CONTROL paragraph. Two magnetic tape files and two mass storage fils are created and verified.

SQ212

X-Numbers: Basic set plus 73.

Features Tested: This program tests all syntactical combinations of the level 2 WRITE statement containing the FROM phrase and ADVANCING mnemonic-name phrase.

SQ213

X-Numbers: Basic set only.

Features Tested: This program tests the level 2 WRITE statement and the File Description for a printer file containing the following LINAGE clause:

```
LINAGE data-name
    FOOTING data-name
    TOP data-name
    BOTTOM data-name
```

The contents of the data-names are changed in order to check the redefinition of logical page formats after page overflow or WRITE ADVANCING PAGE operations.

SQ214

X-Numbers: Basic set only.

Features Tested: This program tests the level 2 WRITE statement and the File Description for a printer file containing the following LINAGE clause:

LINAGE 40
TOP 2

SQ215

X-Numbers: Basic set only.

Features Tested: This program tests the level 2 WRITE statement and the File Description for a printer file containing the following LINAGE clause:

LINAGE IS data-name LINES
TOP 2

The contents of data-name are changed during execution in order to verify the redefinition of logical page formats.

SQ216

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program tests the CLOSE statement with the WITH LOCK phrase. A magnetic tape file is created containing 750 fixed length records of 120 characters. The file is then CLOSEd WITH LOCK and an attempt is made to OPEN the file.

SQ217

X-Numbers: Basic set plus 01, 14, 15, 69, 74, 75, 76, 77.

Features Tested: This program tests the level 2 OPEN file-name series and CLOSE file-name series statements. A magnetic tape file and two mass storage files are created and verified. Each of the files in this program contains 750 fixed length records of 120 characters.

SQ218

X-Numbers: Basic set plus 01, 14, 69, 74, 75, 76.

Features Tested: This program tests the USE AFTER ERROR PROCEDURE for EXTEND and file-name series. A magnetic tape file is created containing 750 fixed length records of 126 characters. The file is then extended by 250 records and the extended file is verified. A mass storage file containing 1000 fixed length records of 126 characters is also created and verified.

SQ219

X-Numbers: Basic set plus 14, 69, 74, 75,

Features Tested: This program tests Sequential I-O operations involving records containing subordinate entries which in turn contain Format 2 OCCURS clauses; i.e., ... OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1 ... . A file (SQ-FS1) is created with 1000 fixed-length 140-character records while varying the value of data-name-1. The file is read back, again while varying the value of data-name-1. The effect of varying the number of occurrences during READ and WRITE operations is evaluated.

SQ220

X-Numbers: Basic set plus 14, 69, 74, 76.

Features Tested: This program tests the CLOSE statement with the LOCK phrase. A mass storage file is created containing 750 fixed length records of 120 characters. The file is then CLOSEd with LOCK and an attempt is made to OPEN the file.

SQ431

X-Numbers: Basic set plus 01, 08, 14, 17, 69, 73, 74, 76, 77, 78, 79, 86.

Features Tested: The program SQ431 contains level 2 Sequential I-O language features for purposes of testing the FIPS PUB 21-1 flagging requirement. The Sequential I-O level 2 elements in this program are contained in High-Intermediate and High levels of Federal Standard COBOL. All Sequential I-O level 2 language features are tested in this program.

D.11 SORT-MERGE MODULE

ST101

X-Numbers: Basic set plus 01, 27, 69, 74, 75.

Features Tested: This program tests the INPUT PROCEDURE and the OUTPUT PROCEDURE of the SORT statement. An output file is created on magnetic tape containing 100 fixed length records of 120 characters each with 10 records per block. Five sort key data items of 1, 2, 3, 4, and 5 characters are defined when the sort file is created and they are tested in the sort output records in the OUTPUT PROCEDURE. The sorted file from program ST101 is passed to ST102 for further processing. The SORT statement in this program is:

```
SORT file-name
     ON ASCENDING KEY ...
     ON DESCENDING KEY ...
     ON ASCENDING KEY ...
     DESCENDING ...   ...
     INPUT PROCEDURE IS ...
     OUTPUT PROCEDURE IS ... THRU ...  .
```

ST102

X-Numbers: Basic set plus 01, 02, 27, 69, 74, 75, 76.

Features Tested: This program tests the USING and GIVING phrases of the SORT statement. The file created in ST101 is used as input by referencing it in the USING phrase. The file created in this program is passed to ST103 for further processing. The output file is referenced in the GIVING phrase. The SORT statement in this program is:

```
SORT file-name
     ON DESCENDING KEY ...
     ON ASCENDING KEY ...
     ON DESCENDING KEY ...
     ASCENDING ...
     USING ...
     GIVING ...   .
```

ST103

X-Numbers: Basic set plus 02, 69, 74, 75.

Features Tested: This program checks the contents of various records and checks for premature end-of-file in the file created by ST102. No SORT statement is present.

ST104

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program builds a 203 record file to be sorted in ST105. Each 18-character record consists of a 6-character SORT key. The file is not created in sorted order.

ST105

X-Numbers: Basic set plus 01, 02, 27, 69, 74, 75, 76.

Features Tested: This program tests the USING phrase and the OUTPUT PROCEDURE of the SORT statement. The file created in ST104 is used as input to the SORT and is referenced in the USING phrase. The SORT statement in this program is:

```
SORT file-name ON
     DESCENDING ...
     key-item
     USING ...
     OUTPUT PROCEDURE ... THRU ...  .
```

ST106

X-Numbers: Basic set plus 01, 27, 69, 74, 75.

Features Tested: This program tests the GIVING phrase and the INPUT PROCEDURE of the SORT statement. An output file is created on magnetic tape containing 9 fixed length records of 27 characters each. Three sort key data items of 1, 8, and 18 characters make up the entire record. After the file is sorted it is passed through the GIVING phrase to ST107 for further processing. The SORT statement in this program is:

```
SORT file-name ON
     ASCENDING ...
     DESCENDING ...
     ASCENDING ...
     INPUT PROCEDURE ... THRU ...
     GIVING ...   .
```

ST107

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program checks the contents of various records in the file created by ST106. No SORT statement is present.

ST108

X-Numbers: Basic set plus 27.

Features Tested: This program tests the use of the INPUT PROCEDURE and the OUTPUT PROCEDURE of the SORT statement and the ability of the compiler to handle eight ascending key data items in one file. The program is completely self-contained. The SORT statement in this program is:

```
SORT file-name ON
     ASCENDING KEY ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     INPUT PROCEDURE ...
     OUTPUT PROCEDURE ... THRU ... .
```

ST109

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program builds a magnetic tape file of 40 variable length records which is sorted in ST110 and checked in ST111. The file is not created in sorted order and consists of records of 50, 75, and 100 characters.

ST110

X-Numbers: Basic set plus 01, 02, 27, 69, 74, 75, 76.

Features Tested: This program tests the USING and GIVING phrases of the SORT statement for a file with variable length records. The file created in ST109 is used as input to the SORT and is referenced in the USING phrase. The sorted output file is named in the GIVING phrase and is passed to ST111 for further processing. Two sort key data items of 10 and 38 characters are defined. The SORT statement in this program is:

```
SORT file-name
     DESCENDING KEY
     key-1
     key-2
     USING ...
     GIVING ... .
```

ST111

X-Numbers: Basic set plus 02, 69, 74, 76.

Features Tested: This program checks the contents of various records in the variable length record file created in ST109 and sorted in ST110. No SORT statement is present in ST111.

ST112

X-Numbers: Basic set plus 06, 69, 74, 79.

Features Tested: This program builds a three reel sequential tape file consisting of 26 records with each record containing 33 identical alphabetic characters varying from A to Z. The file is passed to ST113 where it is sorted and to ST114 where it is tested.

Special Considerations:

(1) A three reel magnetic tape file is required if allowed by the operating system.
(2) The H-option is used to include all CLOSE REEL statements in the source program when a multi-reel file is specified.
(3) The I-option is used to delete all CLOSE REEL statements from the source program when a multi-reel file is not specified.

ST113

X-Numbers: Basic set plus 01, 06, 27, 69, 74, 75, 79.

Features Tested: This program tests the USING and GIVING phrases of the SORT statement for a multi-reel tape file. The three reel tape file created in ST112 is used as input to the SORT and is referenced in the USING phrase. The sorted output file is named in the GIVING phrase and is passed to ST114 as a single reel file. A single sort key data item of 33 characters makes up the entire record. The SORT statement in this program is:

```
SORT file-name DESCENDING .
     key
     USING ...
     GIVING ...  .
```

ST114

X-Numbers: Basic set plus 01, 69, 74, 75.

Features Tested: This program checks the contents of various records in the file created in ST112 and sorted in ST113. The records were

created in ascending order and sorted in descending order.

ST115

X-Numbers: Basic set plus 01, 65, 74, 75.

Features Tested: This program builds a sequential file SQ-FS1 which is passed to ST116 to be sorted. File SQ-FS1 has fixed length records (507 characters/record) and is unblocked. The number of records in this file is set by a four digit integer in X-65. It should be large enough to force the SORT routine in ST116 to be non-core resident; that is, force the system to use some means of auxiliary storage for the SORT sub-strings.

ST116

X-Numbers: Basic set plus 01, 02, 27, 74, 75, 76.

Features Tested: This program is a test of the SORT statement using fixed length records. The collating sequence is native; i.e., no collating statement is used in the actual SORT statement. Qualified alphanumeric and numeric sort keys were used. The actual SORT statement was as follows:

```
SORT sort-file-name
ON ASCENDING KEY key-1 OF data-name-1
ASCENDING key-2 OF data-name-2
  USING file-name-1
  GIVING file-name-2.
```

The input file SQ-FS1 is created in ST115. The output file SQ-FS2 is passed to ST117 for checking.

ST117

X-Numbers: Basic set plus 02, 63, 65, 74, 76.

Features Tested: This program checks the file SQ-FS2 which was sorted into native ascending order by program ST116. X-63 is used to input the system's native ascending collating sequence as a literal of the 51 COBOL characters. Because of confusion over the use of a quote sign imbedded within a non-numeric literal, the quote sign is omitted and the dollar sign is repeated. An example of an X-63 card for a machine with ASCII as the native collating sequence is as follows:

```
X-63   ' $$()*+,-./0123456789;<=>ABCDEFGHIJKLMNOPQRSTUVWXYZ'.
```

The X-65 shows how many records were used in the sequence of programs ST115, ST116, ST117.

ST118

X-Numbers: Basic set plus 27.

Features Tested: This program tests the use of the INPUT PROCEDURE and the OUTPUT PROCEDURE of the SORT statement and the abilty of the compiler to handle eight ascending key data items in one file. Each of the keys identified in the sort statement are elementary data items and use various combinations of picture character-string symbols and clauses for describing the general characteristics of the data item. The program is a rewrite of ST108 and specifically test use of the SIGN clause with sort key fields for the file.

```
SORT file-name ON
     ASCENDING KEY ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     ASCENDING ...
     INPUT PROCEDURE ...
     OUTPUT PROCEDURE ... THRU ...  .
```

ST201

X-Numbers: Basic set plus 01, 14, 27, 28, 29, 69, 74, 75, 76, 77.

Features Tested: This program tests the use of three SORT statements in the same program. Numeric and alphabetic key data items are used. ST201 begins by creating a 100 record file with blocking factor of 10. This file is then sorted GIVING a file with the same file description for use in the second sort. The file is again sorted GIVING a file for use in the third SORT. No file is output from the third SORT. This program also exercises the SAME RECORD AREA clause. The SORT statements in this program are:

```
SORT file-name
     ON DESCENDING KEY ...
     ON ASCENDING KEY ...
     USING ...
     GIVING ... .
```

```
SORT file-name
     ASCENDING ...
     DESCENDING ...
     ASCENDING ...
     INPUT PROCEDURE ...
     GIVING ...  .
```

```
SORT file-name
     ON DESCENDING KEY ...
     ASCENDING ...
     INPUT PROCEDURE IS ...
     OUTPUT PROCEDURE ... THRU ... .
```

ST202

X-Numbers: Basic set plus 06, 27, 28, 74, 75.

Features Tested: This program tests the use of three SORT statements in the same program. The first SORT generates input data in the INPUT PROCEDURE. The SORT results are tested in the OUTPUT PROCEDURE which also creates a two-reel 100 record tape file (via the CLOSE REEL statement) for input to the second SORT. The third SORT exercises the sorted file for the second time. Successful execution is the sole test of this SORT. Alphabetic and numeric sort key data items of 10 characters each are defined. Also tested in this program is the SAME SORT AREA clause. The SORT statements in this program are:

```
SORT file-name
     DESCENDING ...
     INPUT PROCEDURE ...
     OUTPUT PROCEDURE ...  .

SORT file-name
     ASCENDING ...
     USING ...
     OUTPUT PROCEDURE ...  .

SORT file-name
     ASCENDING ...
     USING ...
     OUTPUT PROCEDURE ...  .
```

Special Considerations:

(1) A two-reel magnetic tape file is required if allowed by the operating system.
(2) The H-option is used to include all CLOSE REEL statements in the source program when a multi-reel file is specified.
(3) The I-option is used to delete all CLOSE REEL statements from the source program when a multi-reel file is not specified.

ST203

X-Numbers: Basic set plus 01, 02, 27, 28, 69, 74, 75, 76.

Features Tested: This program tests the use of two SORT statements being

applied to the same 80 character record. The output from the first SORT becomes the input file for the second SORT. The file is copied tape to tape in the mainline section between the sorts, and both copies are checked in the OUTPUT PROCEDURE of the second SORT. The sort key is defined as an 8 character signed numeric data item. This program also tests the use of the SAME SORT AREA clause. The SORT statements in this program are:

```
SORT file-name ON DESCENDING KEY
     key
     INPUT PROCEDURE ...
     GIVING ...   .

SORT file-name ON ASCENDING KEY
     key
     USING ...
     OUTPUT PROCEDURE ...  .
```

ST204

X-Numbers: Basic set plus 01, 02, 27, 69, 74, 75, 76.

Features Tested: This program tests the use of the SORT statement being applied to a file already in sorted order. A magnetic tape of eleven 80-character records is created in an independent section of the program. Then it is sorted employing the INPUT PROCEDURE and OUTPUT PROCEDURE clauses. Finally, the program checks the contents of various records in the file. This program also tests the use of the SAME RECORD AREA clause. The SORT statement in this program is:

```
SORT file-name ON ASCENDING
     key
     INPUT PROCEDURE IS ...
     OUTPUT PROCEDURE IS ...  .
```

ST205

X-Numbers: Basic set plus 01, 02, 03, 04, 27, 69, 74, 75, 76, 77, 78.

Features Tested: This program builds a 21 record file, sorts it, and checks it in the OUTPUT PROCEDURE of the SORT statement. The unusual feature of this program is that the files share a network of SAME AREA, SAME RECORD AREA, and SAME SORT AREA clauses. In order to thoroughly exercise the SAME clause options, two dummy files are opened, closed, read, and written. However, the contents of these two files are never checked. The sort keys are defined as 8 character numeric data items.

ST206

X-Numbers: Basic set plus 01, 27, 69, 74, 75.

Features Tested: This program tests the use of the FROM phrase of the RELEASE statement. Ten records are created with a numerically descending key. The records are then sorted into ascending sequence and the results are spot-checked. The sort keys are defined as 8 character signed numeric data items.

ST207

X-Numbers: Basic set plus 14, 15, 27, 63, 69, 74, 75, 76.

Features Tested: This program is used to test the SORT statement using variable length records which contain an OCCURS DEPENDING ON clause. The input file is a 51 record mass storage file containing records varying in size from 148 to 1435 characters. This program also tests the use of qualified alphanumeric and numeric sort key data items. The SORT statement in this program is:

```
SORT file-name
     ON ASCENDING KEY ... OF ...
     ASCENDING KEY ... OF ...
     USING ...
     GIVING ...  .
```

ST208

X-Numbers: Basic set plus 14, 15, 27, 69, 74, 75, 76.

Features Tested: This program tests the SORT statement using the alphabet-name clause and the COLLATING SEQUENCE phrase to test the ability of the compiler to sort files into the ASCII sequence regardless of the native collating sequence of the machine. Two 51 record mass storage files are initially created to be used in the SORT tests.

Also tested are the use of multiple records in the sort file description, the SAME SORT-MERGE AREA, qualified alphanumeric and numeric sort key data items, and the USING file-name series SORT statement. The SORT statement in this program is:

```
SORT file-name
     ON ASCENDING KEY ... OF ...
     ASCENDING ... OF ...
     COLLATING SEQUENCE IS ...
     USING ...
     GIVING ...  .
```

ST209

X-Numbers: Basic set plus 14, 15, 16, 27, 69, 74, 75, 76, 77.

Features Tested: This proram tests the MERGE statement using the ASCII collating sequence and fixed length records. Two 51 record mass storage files consisting of 132 characters per record are created by the program in ascending ASCII order. Then the alphabet-name clause and the MERGE statement with the COLLATING SEQUENCE phrase are used to test the compiler's ability to merge two files into a third file in ascending ASCII order. This program also tests the use of the SAME SORT-MERGE AREA clause, qualified alphanumeric and numeric merge key data items, and the USING file-name series of the MERGE statement. The MERGE statement in this program is:

```
MERGE file-name
     ASCENDING ... OF ...
     ON ASCENDING KEY ... OF ...
     SEQUENCE ...
     USING ...
     GIVING ...   .
```

ST210

X-Numbers: Basic set plus 08, 14, 15, 27, 69, 74, 75, 76, 77, 78.

Features Tested: This program tests the MERGE statement using the ASCII collating sequence and a multiple file tape. Two files are written on a multiple file tape and a third file is created on mass storage. Then the program tests the compiler's ability to MERGE the second file of the multiple file tape with the mass storage file to produce a new mass storage file. All input files have 51 fixed length records with 132 characters per record. The MERGE file should have 102 records. This program also tests the SAME SORT AREA clause, qualified alphanumeric and numeric merge key data items, USING file-name series, and RETURN RECORD INTO statement. The MERGE statement in this program is:

```
MERGE file-name
     ON DESCENDING KEY ... OF ...
     ASCENDING ... OF ...
     COLLATING SEQUENCE IS ...
     OUTPUT PROCEDURE IS ...
```

ST211

X-Numbers: Basic set plus 14, 15, 16, 27, 69, 74, 75, 76, 77.

Features Tested: This program tests the MERGE statement using the COLLATING SEQUENCE phrase and a user-defined alphabet. Also tested is the SORT statement with the COLLATING SEQUENCE phrase

and the alphabet-name IS STANDARD-1. Two 51 record mass storage files are created. Then they are merged according to the user defined sequence to produce a 102 record mass storage file. Finally, the file is sorted into ascending ASCII sequence by the SORT statement. All files consist of 132 character records. This program also tests the SAME SORT AREA clause, qualified merge key data items, USING file-name series clause, and the RETURN ... INTO ... statement.

```
MERGE file-name
     ON ASCENDING KEY ... OF ... ,  ... OF ...
     COLLATING SEQUENCE IS ...
     USING ... , ...
     OUTPUT PROCEDURE IS ...  .
```

ST212

X-Numbers: Basic set plus 14, 15, 27, 63, 69, 74, 75, 76.

Features Tested: This program tests the SORT statement using fixed length records. Two 51 record mass storage files are created, then they are input to the SORT and the GIVING phrase rewrites the first file. All files consist of 132 character records. The SORT statement in this program is:

```
SORT file-name
     ON ASCENDING KEY ... OF ...
     ASCENDING ... OF ...
     USING ... , ...
     GIVING ...
```

ST213

X-Numbers: Basic set plus 14, 15, 16, 27, 64, 69, 74, 75, 76, 77.

Features Tested: This program tests the MERGE statement using a native collating sequence and fixed length records. Initially, two 51 record mass storage files are created in descending native order. The MERGE statement is used to merge the two files into a third file of 102 records in descending native collating sequence. All files consist of 132 character records. The MERGE statement in this program is:

```
MERGE file-name
     DESCENDING ... OF ...
     ON DESCENDING KEY ... OF ...
     USING ...
     GIVING ...  .
```

ST214

X-Numbers: Basic set plus 08, 14, 15, 64, 74, 75, 76, 77, 78.

Features Tested: This program tests the MERGE statement using a native collating sequence and a multiple file tape. Initially, two 51 record files are written on a multiple file tape in ascending native order. Then a third file is written on mass storage. Finally, the MERGE statement uses the second and third files to create a new 102 record mass storage file. All files consist of 132 character records. The MERGE statement in this program is:

```
MERGE file-name .
     ON DESCENDING KEY ... OF ...
     ASCENDING ... OF ...
     USING ...
     OUTPUT PROCEDURE IS ...   .
```

ST215

X-Numbers: Basic set plus 14, 15, 16, 27, 63, 69, 74, 75, 76, 77.

Features Tested: This program tests the MERGE statement using the COLLATING SEQUENCE phrase and a user defined alphabet. The SORT statement without the COLLATING SEQUENCE specified is also tested. In this case the native sequence is used. Two 51 record tape files are initially created. Then the MERGE statement combines these files to form a third mass storage file with 102 records in ascending order. This file is then sorted into ascending native sequence by the SORT statement giving back the first file with 102 records. This program contains the following MERGE and SORT statements:

```
MERGE file-name
     ON ASCENDING KEY ... OF ... , ... OF ...
     COLLATING SEQUENCE IS ...
     USING ...
     OUTPUT PROCEDURE IS ...  .

SORT file-name
     ON ASCENDING KEY ... OF ...
     ASCENDING ... OF ...
     USING ...
     GIVING ...  .
```

ST216

X-Numbers: Basic set plus 14, 15, 27, 69, 74, 75, 76

Features Tested: This program tests SORT operations involving records containing subordinate entries which in turn contain Format 2 OCCURS clauses; i.e. ... OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1 ... . A file (SQ-FS1) is

created with 1000 fixed-length 140-character records. The file is then SORTed, RELEASEing and RETURNing records using record descriptions with the Format 2 OCCURS clause. The effect of varying data-name-1 is evaluated; the performance of the SORT verb is none checked.

ST431

X-Numbers: Basic set plus 14, 27, 74, and 75.

Features Tested: The program ST431 contains level 1 Sort/Merge features for purposes of testing the FIPS PUB 21-1 flagging requirement. The SRT level 1 elements in this program are contained in the High-Intermediate and High levels of Federal Standard COBOL. The Data Division clause to be flagged is:

SD file-name.

The Procedure Division statements to be flagged are:

SORT file-name.

RELEASE sort-record-name.

ST432

X-Numbers: Basic set plus 14, 15, 27, 74, 75, and 76.

Features Tested: The program ST432 contains level 1 Sort/Merge features for purposes of testing the FIPS PUB 21-1 flagging requirement. The SRT level 1 elements in this program are contained in the High-Intermediate and High levels of Federal Standard COBOL. The Data Division clause to be flagged is:

SD file-name.

The Procedure Division statements to be flagged are:

SORT file-name.

RETURN sort-name RECORD.

ST433

X-Numbers: Basic set plus 1, 15, 27, 74, 76, and 77.

Features Tested: The program ST433 contains level 1 Sort/Merge features for purposes of testing the FIPS PUB 21-1 flagging requirement. The SRT level 1 elements in this program are contained in the High-Intermediate and High levels of Federal Standard COBOL. The Data Division clause to be flagged is:

SD file-name.

The Procedure Division statement to flagged is:

SORT sort-file-name.

ST434

X-Numbers: Basic set plus 1, 74, and 77.

Features Tested: The program ST434 tests the file produced in program ST433 with respect to the final sort sequence of the file generated and sorted in ST431, sorted in ST432 and also in ST433.

No elements of this program should be flagged.

ST441

X-Numbers: Basic set plus 1, 14, 15, 27, 74, 75, 76, and 77.

Features Tested: The program ST441 contains level 2 Sort/Merge features for purposes of testing the FIPS PUB 21-1 flagging requirement. The SRT level 2 elements in this program are contained in the High level of Federal Standard COBOL. The Environment Division clause to be flagged is:

SAME SORT AREA sort-file-name, file-name.

The Procedure Division statements to be flagged are:

SORT USING file-name series.

Multiple SORTs in the Procedure Division.

ST442

X-Numbers: Basic set plus 2, 14, 15, 27, 74, 75, 76, and 78.

Features Tested: The program ST442 contains level 2 Sort/Merge features for purposes of testing the FIPS PUB 21-1 flagging requirement. The SRT level 2 elements in this program are contained in the High level of Federal Standard COBOL. The Environment Division clause to be flagged is:

SAME RECORD AREA FOR file-name, sort-file-name.

The Procedure Division statements to be flagged are:

SORT USING file-name series.

Multiple SORTs in the Procedure Division.

COLLATING SEQUENCE phrase in the SORT statement.

ST443

X-Numbers: Basic set plus 1, 2, 14, 15, 27, 74, 75, 76, 77, and 78.

Features Tested: The program ST443 contains level 2 Sort/Merge features for purposes of testing the FIPS PUB 21-1 flagging requirement. The SRT level 2 elements in this program are contained in the High level of Federal Standard COBOL. The Environment Division clause to be flagged is:

SAME SORT-MERGE AREA FOR file-name, sort-file-name.

The Procedure Division statements to be flagged are:

MERGE merge-file-name.

Multiple MERGEs in the Procedure Division.

D.12 TABLE HANDLING MODULE

TH101

X-Numbers: Basic set only.

Features Tested: This program tests the use of the format 1 SET statement. Various combinations of setting identifiers and index-names to identifiers, index-names, and integers are tested.

TH102

X-Numbers: Basic set only.

Features Tested: This program tests the use of numeric literal subscripts to reference internal table items.

TH103

X-Numbers: Basic set only.

Features Tested: This program tests the use of subscripts, composed of numeric integer data-names, to reference internal table items.

TH104

X-Numbers: Basic set only.

Features Tested: This program tests the use of the SET statement in conjunction with a redefined table. Index-names are SET to numeric integers, USAGE index data items, and indices assigned to other tables.

TH105

X-Numbers: Basic set only.

Features Tested: This program tests the accessing of a three dimensional table using numeric literals and data-names as subscripts.

TH106

X-Numbers: Basic set only.

Features Tested: This program tests the use of index-names to reference tables that have been redefined, omitting the INDEXED BY clause. Also tested are the SET ... UP BY ... and SET ...DOWN BY ... statements.

TH107

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using various combinations of numeric literal subscripts.

TH108

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using various combinations of internal indices.

TH109

X-Numbers: Basic set only.

Features Tested: This program tests the use of spaces, commas, and left and right parentheses as separators in statements which reference table items.

TH110

X-Numbers: Basic set only.

Features Tested: This program tests the use of a mixture of literals and index-names to reference table items.

TH111

X-Numbers: Basic set only.

Features Tested: This program tests the use of the SET ... UP BY ... and SET .. DOWN BY ... statements using various combinations of positive, negative, zero, and unsigned numeric literals and data-names as indices.

TH151

X-Numbers: Basic set only.

Features Tested: This program tests the use of numeric literal subscripts to reference internal table items.

TH152

X-Numbers: Basic set only.

Features Tested: This program tests the use of subscripts composed of numeric integer data-names to reference internal table items.

TH201

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing and subscripting to reference individual table items. This program tests the use of the format 1 SEARCH statement with VARYING and AT END phrases and the format 1 SET statement.

TH202

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing and subscripting to reference individual table items. This program tests the use of the format 1 SEARCH statement with VARYING and AT END phrases and the format 1 SET statement.

TH203

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing and subscripting to reference individual table items. This program tests the use of the format 1 SEARCH statement with VARYING and AT END phrases and the format 1 SET statement.

TH204

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing and subscripting to reference individual table items. This program tests the use of the format 1 SEARCH statement with VARYING and AT END phrases and the format 1 SET statement.

TH205

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing and subscripting. The table is defined using the OCCURS, ASCENDING KEY and INDEXED BY clauses. This program tests the use of the format 2 SEARCH statement with the AT END phrase and the format 1 SET statement.

TH206

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing and subscripting. The table is defined using the OCCURS, ASCENDING KEY and INDEXED BY clauses. This program tests the use of the format 2 SEARCH statement with the AT END phrase and a single WHEN phrase, and a format 1 SET statement.

TH207

X-Numbers: Basic set only.

Features Tested: This program verifies the accuracy in building and accessing a three dimensional table using indexing. The table is defined using the OCCURS and INDEXED BY clauses and then it is redefined with a second table. This program tests the use of the format 1 SEARCH statement with the VARYING and AT END phrases, and the format 1 SET statement.

TH208

X-Numbers: Basic set only.

Features Tested: This program defines a one dimensional table which has a variable number of occurrences of table items and then references the items using indexing. The table is defined using the format 2 OCCURS clause and consists of several condition-name entries. The program tests the use of the format 1 and 2 SEARCH statements.

TH209

X-Numbers: Basic set only.

Features Tested: This program defines a one dimensional table which redefines an elementary item. The program then tests the use of the format 1 SET statement and the format 1 SEARCH statement with VARYING and AT END clauses. An index assigned to one table or an index data item is then varied when searching the second table. A comparison is made to verify that the indices

or the index and index data items are varied properly.

TH210

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table containing various ascending and descending keys that are referenced by subscripts and indices which have computational values or have been set to computational items. Some of the optional words have been eliminated from the syntax.

TH211

X-Numbers: Basic set only.

Features Tested: This program defines a two dimensional table containing various ascending and descending keys and multiple indices. The program then tests the use of the format 1 and 2 SEARCH statements with and without multiple conditions in the WHEN phrase to reference the table.

TH212

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table and tests the reference of individual table items using indexing.

TH213

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table and tests the reference of individual table items using signed and unsigned numeric items as subscripts.

TH214

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table and tests the reference of individual table items using unsigned numeric items as indices and subscripts.

TH215

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table and tests the reference of individual table items using indexing.

TH216

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table and tests the reference of individual table items using indexing and subscripting.

TH217

X-Numbers: Basic set only.

Features Tested: This program defines a three dimensional table and tests the reference of individual table items using indexing and subscripting.

TH218

X-Numbers: Basic set only.

Features Tested: This program defines a two dimensional table containing the OCCURS clause with the INDEXED BY series clause. The program tests relative addressing and various combinations of the format 2 SET statement.

TH219

X-Numbers: Basic set only.

Features Tested: This program defines a two dimensional and a three dimensional table and tests the use of the semicolon as a separator in referencing table items via subscripting and indexing.

TH220

X-Numbers: Basic set only.

Features Tested: This routine defines several tables which must be qualified in order to reference the table items. The use of qualified subscripts in referencing both unqualified and qualified table items is tested. Additionally, there are conditional statements which reference qualified condition-names with both unqualified and qualified subscripts.

TH221

X-Numbers: Basic set only

Features Tested: This program tests various Nucleus module verbs acting on WORKING-STORAGE records containing subordinate entries which in turn contain Format 2 OCCURS clauses; i.e. ... OCCURS integer-1 TO integer-2 TIMES DEPENDING ON data-name-1 ... . The tests ascertain whether the compiler treats the number of table occurrences as variable depending on the current value of the data-name-1 identifier. Operations tested include IF, INSPECT, MOVE, SEARCH, STRING, and UNSTRING.

TH431

X-Numbers: Basic set only.

Features Tested: The program TH431 contains the level 2 Table Handling features for purposes of testing the FIPS PUB 21-1 flagging requirement. The TBL level 2 elements in this program are contained in the High-Intermediate Level of Federal Standard COBOL. Data Division clauses to be flagged are

```
OCCURS DEPENDING ON
ASCENDING KEY
DESCENDING KEY.
```

The Procedure Division statements to be flagged are

```
SEARCH identifier.
SEARCH ALL identifier.
```

E. SYSTEM DEPENDENT INFORMATION FOR PPLVP74 IMPLEMENTATION

There may be some modifications necessary to the source code for the PPLVP74 processor beyond those identified in Appendix B. Through the use of the "Bootstrap" program shown in Appendix B certain modifications may be made during the process of extracting PPLVP74 from the Population File (source program library). There are several places in the source code of PPLVP74 where comment entries have been included to facilitate certain limited modifications which can be made through the use of the Bootstrap program. [The bootstrap program could be used to merely extract the source program for further modification through the use of system utilities (e.g., text editors, source program update processor).]

The referenced comment entries are all unique and are located in

1. The File Description entry for the Population File.

2. The File Description entry for the Source Program File.

3. The Read routine for the Population File following the READ statement. The record is available by referencing the record name SOURCE-IN-2400.

4. The Read routine for the Source Program File following the READ statement. The record is available by referencing the record name UD-RECORD.

Source code can be inserted at these four locations in the PPLVP74 source program by making modifications to the Bootstrap program. The section named WRITE-CARD contains the statement which writes the current source image of PPLVP74 to the file from which PPLVP74 will be further processed. By testing for the data-name VP-12-18 against one of the values of the comment entries, the comment entry can be located. When an equal compare exists then the appropriate code can be inserted.

Example

1. The UNIVAC EXEC-8 Field Data COBOL compiler requires that the File Description entry for the Population File must contain a nonstandard clause in order to read the magnetic tape as produced by the Software Development Division. This clause, 'RECORDING MODE 1', can be inserted by the following source code being placed in 'WRITE-CARD' before the WRITE statement for VP-REC.

```
      IF  VP-12-18 EQUAL 'FDPOPFI'
          WRITE VP-REC
          MOVE 'RECORDING MODE 1' TO VP-12-72.
*     The statement WRITE VP-REC will follow immediately
*     and cause the statement to be inserted.
```

2. The HONEYWELL GCOS COBOL compiler requires that the File Description entry for the Population File must contain a nonstandard clause in order to read the magnetic tape as produced by the Software Development Division. This

clause, 'RECORDING MODE BCD', can be inserted by the following source code being placed in 'WRITE-CARD' before WRITE statement for VP-REC.

```
IF  VP-12-18 EQUAL TO 'FDPOPFI'
    WRITE VP-REC
    MOVE 'RECORDING MODE BCD' TO VP-12-72.
```

3. A modified Bootstrap Source Program for the IBM OS/VS operating systems follows. The modifications are more complex and as such are shown in their complete context. The character set (special characters) must be converted both while the PPLVP74 processor is being extracted and when the PPLVP74 processor extracts source programs for the Population File. The following source code must be inserted in Read routine for the Population File:

```
IF  BCD-POP EQUAL TO 'YES'
    TRANSFORM SOURCE-IN-2400 FROM '@' TO QUOTE
    TRANSFORM SOURCE-IN-2400 FROM '<+&=#%' TO ')<+>=('.
```

The following source code must be inserted in the Read routine for the CONTROL-CARD file:

```
IF BCD-CTL EQUAL TO 'YES'
   TRANSFORM UD-RECORD FROM '@' TO QUOTE
   TRANSFORM UD-RECORD FROM '<+&=#%' TO ')<+>=('.
```

The Bootstrap Program for OS/VS follows.

```
//UTBASVP8  JOB  (T99905,CC10),'FCCTS',MSGLEVEL=1,CLASS=B,
// MSGCLASS=A
//STEPU EXEC COBVCLG
//COB.SYSIN DD *
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           IBMOSB.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT    POPFILE ASSIGN TO
               UT-S-SYSUT1.
           SELECT  PRT ASSIGN TO
               UT-S-SYSOUT.
           SELECT  VP-FILE ASSIGN TO
               UT-S-SYSUT2.
       DATA DIVISION.
       FILE SECTION.
       FD  POPFILE
           LABEL RECORDS OMITTED
           DATA RECORD POP-REC
           BLOCK CONTAINS 1 RECORDS
           RECORD CONTAINS 2400 CHARACTERS.
           01  POP-REC.
```

```
    02 CARD1 PIC X(80) OCCURS 30 TIMES.
FD  VP-FILE
    LABEL RECORDS STANDARD
    DATA RECORD IS VP-REC.
01  VP-REC.
    02 VP-1-72.
        05 VP-1-6 PIC X(6).
        05 VP-7-7 PIC X.
        05 VP-8-11 PIC X(4).
        05 VP-12-72.
            07 VP-12-18.
            09 VP-12-16 PIC XXXXX.
            09 VP-17-19 PIC 99.
            88 X01 VALUE 001.
            88 X02 VALUE 002.
            88 X03 VALUE 003.
            88 X04 VALUE 004.
            88 X10 VALUE 010.
            88 X11 VALUE 011.
            88 X12 VALUE 012.
            88 X13 VALUE 013.
            88 X14 VALUE 014.
            88 X15 VALUE 015.
            88 X16 VALUE 016.
            88 X17 VALUE 017.
            88 X18 VALUE 018.
            88 X19 VALUE 019.
            88 X36 VALUE 036.
            88 X37 VALUE 037.
            88 X45 VALUE 045.
            88 X48 VALUE 048.
            88 X49 VALUE 049.
            88 X50 VALUE 050.
            07 FILLER PIC X(54).
    02 VP-73-77 PIC X(5).
    02 VP-78-80 PIC X(3).
FD  PRT
    LABEL RECORDS STANDARD
    DATA RECORD PRT-REC.
01  PRT-REC.
        02 FILLER PIC X(130).
WORKING-STORAGE SECTION.
01  SUB1 PIC S9(9) USAGE COMP SYNC RIGHT VALUE 55.
01  VP-REC-HOLD PIC X(80).
*COMMENT 1
*     The following data item contains the implementor names and
*     table values to be used as the PPLVP74 processor is generated
*     See Appendix B for a definition of what is contained in each
*     data item.
01  CONTROL-CARDS.
    03 CTL-01 PIC X(28) VALUE 'UT-S-SYSUT1.          '.
    03 CTL-02 PIC X(28) VALUE 'UT-S-SYSUT2.          '.
    03 CTL-03 PIC X(28) VALUE 'UT-S-SYSUT3.          '.
    03 CTL-04 PIC X(28) VALUE 'UT-S-SYSUT4.          '.
    03 CTL-10 PIC X(28) VALUE '58.                   '.
```

```
      03 CTL-11 PIC X(28) VALUE '58.                          '.
      03 CTL-12 PIC X(28) VALUE '58.                          '.
      03 CTL-13 PIC X(28) VALUE '10.                          '.
      03 CTL-14 PIC X(28) VALUE '30.                          '.
      03 CTL-15 PIC X(28) VALUE '50.                          '.
      03 CTL-16 PIC X(28) VALUE '10.                          '.
      03 CTL-17 PIC X(28) VALUE '150.                         '.
      03 CTL-18 PIC X(28) VALUE '10.                          '.
      03 CTL-19 PIC X(28) VALUE '200.                         '.
      03 CTL-36 PIC X(28) VALUE 'UT-S-SYSOUT.              '.
      03 CTL-37 PIC X(28) VALUE 'UT-S-SYSIN.            '.
      03 CTL-45 PIC X(28) VALUE 'C01                        '.
      03 CTL-48 PIC X(28) VALUE 'OMITTED                  '.
      03 CTL-49 PIC X(28) VALUE 'IBM-360.                '.
      03 CTL-50 PIC X(28) VALUE 'IBM-360.                      '.
*COMMENT 2
*     The following working storage records (LINE-1 through LINE-6)
*     represent source statements which are to be inserted into
*     PPLVP74 as it is extracted from the Population File.  The
*     complete text contained in these records is shown in the
*     narrative describing the program.
  01  LINE-1.
      02 FILLER PIC X(28) VALUE
      'IF        BCD-POP EQUAL TO  '.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE XXX VALUE 'YES'.
      02 FILLER PICTURE X VALUE QUOTE.
  01  LINE-2.
      02 FILLER PICTURE X(39) VALUE
      '          TRANSFORM SOURCE-IN-2400 FROM '.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE X VALUE '@'.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE X(11) VALUE ' TO QUOTE'.
  01  LINE-3.
      02 FILLER PICTURE X(39) VALUE
      '          TRANSFORM SOURCE-IN-2400 FROM '.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE X(06) VALUE '<+&=#%'.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE XXXX VALUE ' TO '.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE X(06)  VALUE ')<+>=('.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE X VALUE '.'.
  01  LINE-4.
      02 FILLER PIC X(28) VALUE
      'IF         BCD-CTL EQUAL TO '.
      02 FILLER PICTURE X VALUE QUOTE.
      02 FILLER PICTURE XXX VALUE 'YES'.
      02 FILLER PICTURE X VALUE QUOTE.
  01  LINE-5.
      02 FILLER PICTURE X(39) VALUE
      '          TRANSFORM UD-RECORD FROM    '.
      02 FILLER PICTURE X VALUE QUOTE.
```

```
     02 FILLER PICTURE X VALUE '@'.
     02 FILLER PICTURE X VALUE QUOTE.
     02 FILLER PICTURE X(11) VALUE ' TO QUOTE'.
 01  LINE-6.
     02 FILLER PICTURE X(39) VALUE
     '          TRANSFORM UD-RECORD FROM   '.
     02 FILLER PICTURE X VALUE QUOTE.
     02 FILLER PICTURE X(06) VALUE '<+&=#%'.
     02 FILLER PICTURE X VALUE QUOTE.
     02 FILLER PICTURE XXXX VALUE ' TO '.
     02 FILLER PICTURE X VALUE QUOTE.
     02 FILLER PICTURE X(06)  VALUE ')<+>=('.
     02 FILLER PICTURE X VALUE QUOTE.
     02 FILLER PICTURE X VALUE '.'.
 PROCEDURE DIVISION.
 DRIVER-SECTION SECTION.
 DRIVER1.
     OPEN    INPUT POPFILE.
     OPEN    OUTPUT PRT VP-FILE.
     MOVE 'START' TO PRT-REC.
     WRITE PRT-REC.
*    *** DOWHILE VP-1-72 NOT EQUAL TO . . .
     PERFORM READ-POP UNTIL VP-1-72 EQUAL TO
             '*HEADER,COBOL,ASVP9   '.
     WRITE    PRT-REC FROM VP-REC.
*    *** DOUNTIL VP-1-6 EQUAL TO . . .
     PERFORM SELECT-VP.
     WRITE    PRT-REC FROM VP-REC.
     PERFORM SELECT-VP UNTIL
         VP-1-6 EQUAL TO '*END-O' OR
         VP-1-6 EQUAL TO '*HEADE'.
     WRITE    PRT-REC FROM VP-REC.
     MOVE  'FIN   ' TO PRT-REC.
     WRITE PRT-REC.
     CLOSE PRT VP-FILE POPFILE.
     STOP RUN.
 SELECT-VP SECTION.
 SLE-01.
     PERFORM READ-POP.
     IF  VP-1-6 EQUAL TO '*HEADE' OR
         VP-1-6 EQUAL TO '*FILES' OR
         VP-1-6 EQUAL TO '*END-O'
             GO TO SELECT-EXIT
     ELSE
         PERFORM PROCESS-VP
         IF  VP-7-7 EQUAL TO ' ' OR
             VP-7-7 EQUAL TO '-' OR
             VP-7-7 EQUAL TO '*'
             PERFORM WRITE-CARD
         ELSE
             IF VP-7-7 EQUAL TO 'A' OR
                VP-7-7 EQUAL TO 'E' OR
                VP-7-7 EQUAL TO 'C' OR
                VP-7-7 EQUAL TO 'L' OR
                VP-7-7 EQUAL TO 'I'
```

```
                MOVE SPACE TO VP-7-7
                PERFORM WRITE-CARD.
*           ENDIF
*       ENDIF
*   ENDIF
 SELECT-EXIT.
 READ-POP SECTION.
 READ1.
     IF       SUB1 GREATER 30
              MOVE  1  TO SUB1
              PERFORM READP1.
     MOVE CARD1 (SUB1) TO VP-REC
*COMMENT 3
*    The following TRANSFORM statements will convert the special
*    characters on the population file to the appropriate EBCDIC
*    character required by the 360/370.
     TRANSFORM VP-12-72 FROM '@' TO QUOTE.
     TRANSFORM VP-12-72 FROM '<+&=#%' TO ')<+>=('.
     IF      VP-1-6 EQUAL TO '*HEADE'
     WRITE     PRT-REC FROM VP-REC.
     ADD  1 TO SUB1.
 READP1 SECTION.
 RDD.
     READ POPFILE AT END MOVE '*END-O' TO VP-1-6.
 PROCESS-VP SECTION.
 PROC1.
     IF       VP-12-16 EQUAL TO 'XXXXX' AND
         VP-17-19     NUMERIC
              NEXT SENTENCE ELSE GO TO PROCEND.
     IF       X01 MOVE  CTL-01 TO VP-12-72 GO TO PROCEND.
     IF       X02 MOVE  CTL-02 TO VP-12-72 GO TO PROCEND.
     IF       X03 MOVE  CTL-03 TO VP-12-72 GO TO PROCEND.
     IF       X04 MOVE  CTL-04 TO VP-12-72 GO TO PROCEND.
     IF       X10 MOVE  CTL-10 TO VP-12-72 GO TO PROCEND.
     IF       X11 MOVE  CTL-11 TO VP-12-72 GO TO PROCEND.
     IF       X12 MOVE  CTL-12 TO VP-12-72 GO TO PROCEND.
     IF       X13 MOVE  CTL-13 TO VP-12-72 GO TO PROCEND.
     IF       X14 MOVE  CTL-14 TO VP-12-72 GO TO PROCEND.
     IF       X15 MOVE  CTL-15 TO VP-12-72 GO TO PROCEND.
     IF       X16 MOVE  CTL-16 TO VP-12-72 GO TO PROCEND.
     IF       X17 MOVE  CTL-17 TO VP-12-72 GO TO PROCEND.
     IF       X18 MOVE  CTL-18 TO VP-12-72 GO TO PROCEND.
     IF       X19 MOVE  CTL-19 TO VP-12-72 GO TO PROCEND.
     IF       X36 MOVE  CTL-36 TO VP-12-72 GO TO PROCEND.
     IF       X37 MOVE  CTL-37 TO VP-12-72 GO TO PROCEND.
     IF       X45 MOVE  CTL-45 TO VP-12-72 GO TO PROCEND.
     IF       X48 MOVE  CTL-48 TO VP-12-72 GO TO PROCEND.
     IF       X49 MOVE  CTL-49 TO VP-12-72 GO TO PROCEND.
     IF       X50 MOVE  CTL-50 TO VP-12-72 GO TO PROCEND.
 PROCEND. EXIT.
 WRITE-CARD SECTION.
 SWRITE.
*COMMENT 4
*    The following two IF statements locate the READ routine for both
*    the Control Card File and the Population File.  The source code
```

```
      *     to be inserted is described under comment 2.  The results of
      *     the modification permit the user to control whether the conver-
      *     sion of special characters will be done for either the Control-
      *     Card File or the Population File.  This would be handled through
      *     the following two PPLVP74 control statements:
      *
      *         1.  *BCDPOP     YES
      *         2.  *BCDPOP     NO
      *
      *         1.  *BCDCTL     YES
      *         2.  *BCDCTL     NO
      *
      *     The YES options would cause character conversion.
      *     The NO options would inhibit character conversion.
            IF       VP-12-18 EQUAL TO 'RDPOPFI'
                      WRITE VP-REC
                      MOVE  SPACE TO VP-REC
                      MOVE  LINE-1 TO VP-12-72
                      WRITE VP-REC
                      MOVE  SPACE TO VP-REC
                      MOVE  LINE-2 TO VP-12-72
                      WRITE VP-REC
                      MOVE SPACE TO VP-REC
                      MOVE LINE-3 TO VP-12-72.
            IF       VP-12-18 EQUAL TO 'RDCTLFI'
                      WRITE VP-REC
                      MOVE SPACE TO VP-REC
                      MOVE LINE-4 TO VP-12-72
                      WRITE VP-REC
                      MOVE SPACE TO VP-REC
                      MOVE LINE-5 TO VP-12-72
                      WRITE VP-REC
                      MOVE SPACE TO VP-REC
                      MOVE LINE-6 TO VP-12-72.
            WRITE     VP-REC.
/*
//60.SYSUT1 DD  DSN=&&SYSUT1,
//             UNIT=TAPE7,
//             LABEL=(,BLP),
//             DISP=(OLD,KEEP),
//   DCB=(DEN=2,TRTCH=ET),
//             VOL=SER=15456
SYSUT1 DD  DSN=HOLMES.INPT.SOURCE,DISP=SHR,UNIT=3330,
VOL=SER=USER04
//GO.SYSUT2 DD  DSN=&ASVP8,
//             UNIT=SYSDA,
//             SPACE=(460,(700,100)),
//             DCB=(LRECL=80,BLKSIZE=400,RECFM=FB),
//             DISP=(NEW,PASS)
//GO.SYSOUT DD  SYSOUT=A
//STEP2  EXEC  COBVCL,
//             COND.LKED=(9,LT)
//COB.SYSIN  DD DSN=&ASVP8,
//             UNIT=SYSDA,
//             DISP=(OLD,DELETE)
```

```
//LKED.SYSLMOD DD DISP=SHR,DSN=HOLMES.INPT.LOAD(ASVP8),
//  UNIT=3330,VOL=SER=USER04
/*
//
//
```

The following discussion describes the philosophy of the implementation of the VP-routine. The major output file of the VP-routine (Source-Output file) is where the source programs will be directed. The minimum degree of sophistication that the file should attain is to contain source programs made syntactically correct by the VP-routine. (The use of only the 'X-cards' described in the documentation.) The maximum degree of sophistication of this file will be syntactically correct source programs and all necessary Operating System Control Statements to establish a complete job or run, allocate any necessary files, compile and execute each source program. For the three systems mentioned above:

a. EXEC 8 - The output file is in SDF format (alternate punch file) which can be added to the run stream. The added file automatically starts the generated runs.

b. GCOS - The output file is in System Standard Format and is an IMCV tape containing one or more SNUMBs.

c. OS - The output file can be introduced as a reader and contains one or more jobs.

The conversion of the data on the Population File is unnecessary for the three systems. Character/code conversion can be handled automatically:

a. EXEC 8 - The RECORDING MODE 1 clause in the source program and the use of the "HIE" options in the assign (@ASG) card for the Population File.

b. GCOS - The RECORDING MODE BCD clause in the source program is all that is necessary in that the $ INCODE control statement is generated on the IMCV tape for transliteration of those characters which must be converted.

c. OS - The DD statement for the 7 track Population File should have the following DCB option "DCB=(DEN=2,TRTCH=ET)" to convert the 6 bit characters to 8 bit characters. The VP-routine will handle the conversion of special characters through the use of the TRANSFORM statements added to the source program. Prior to any *ENVIRONMENT statement a "BCD-POP NO" should appear so that no transformation of the special characters in the VP-routine control cards from the population file takes place. Prior to the *END-MONITOR card a "BCD-POP YES" should appear so that the source programs extracted from the population file will have all special characters translated to the appropriate EBCDIC characters.

NOTE: The format for the *BCD-POP card is:

```
*BCD-POP    YES
*BCD-POP    NO
```

F. Corrections for Version 3 Release 0 as of 77/09/01

This appendix represents Temporary Program Fixes (TPF) which should be applied to the CCVS. These errors were discovered too late to be included in the Population file for version 3.0.

Temporary Program Fix Number:

Program Name:

Problem Description:

(No errors have been discovered.)

BIBLIOGRAPHIC DATA SHEET	1. Report No. FCCTS/CCVS74-77/20	2.	3. Recipient's Accession No.
4. Title and Subtitle 1974 COBOL Compiler Validation System Version 3.0 User's Guide (Implementation Documentation)			5. Report Date October 1977 6.
7. Author(s) See 9.			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Federal COBOL Compiler Testing Service Department of the Navy Washington, DC 20376			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address Federal COBOL Compiler Testing Service Department of the Navy Washington, DC 20376			13. Type of Report & Period Covered 14.

15. Supplementary Notes
Supercedes and replaces ADA036174

16. Abstracts
This document represents the implementation information necessary to implement Version 3.0 of the 1974 U. S. Navy COBOL Compiler Validation System (CCVS). The 1974 CCVS will consist of audit routines, their related data, and an executive routine (VP-routine) which prepares the audit routines for compilation. Each audit routine is a COBOL program which includes many tests and supporting procedures indicating the result of the tests. The audit routines collectively contain all of the features of American National Standard Programming Language COBOL - X3.23-1974 (except for the ENTER statement and arithmetic expressions in the Nucleus module and the Communications module in its entirety) as specified in Federal Information Processing Standard (FIPS) 21-1.
The COBOL Compiler Validation System tape is available from NTIS as report number FCCTS/CCVS74-77/19.

17. Key Words and Document Analysis. 17a. Descriptors

COBOL
Verifying
Software
Audit Routines
Validation
Compilers
Standards
Programming Languages
Portability

17b. Identifiers/Open-Ended Terms

CCVS (COBOL Compiler Validation System)
CVS (Compiler Validation System)

ACCESSION for
NTIS White Section X
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A 21

17c. COSATI Field/Group 09/02

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
RELEASE UNLIMITED	UNCLASSIFIED	
	20. Security Class (This Page) UNCLASSIFIED	22. Price